Entwurf und Implementierung eines OpenPGP Clients

Diplomarbeit von Carsten Längsfeld

27. November 2007

Technische Universität Darmstadt
Fachbereich Informatik
Institut für Kryptographie und Computeralgebra

Betreuer: Dr. Vangelis Karatsiolis

Prüfer: Prof. Johannes Buchmann

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 27. November 2007 _____

Carsten Längsfeld

Danksagung

Mein ganz besonderer Dank geht an Caren Beber, Eva Kunze, Radostina Boneva, Stephan Pravida und Björn Kresse für Korrekturlesen und mentale Unterstützung. Weiterhin möchte ich mich bei Dr. Vangelis Karatsiolis für die Betreuung dieser Arbeit bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

Tabellenverzeichnis **viii**

1 Einleitung **1**
1.1 Aufgabenstellung 1
1.2 Gliederung der Arbeit 2

2 Kryptographische Grundlagen **4**
2.1 Verschlüsselung 5
2.2 Hash-Funktionen 8
2.3 Digitale Signaturen 9

3 Public-Key-Infrastrukturen **11**
3.1 Hierarchisch organisierte Public-Key-Infrastruktur 11
3.2 Netz des Vertrauens (Web of Trust) 14
3.2.1 Direkte Validierung 15
3.2.2 Validierung über das Netz des Vertrauens 16
3.3 Probabilistisches Vertrauensmodell 18

4 OpenPGP Nachrichten Format **22**
4.1 Aufbau von OpenPGP Nachrichten 23
4.1.1 Pakete im alten Format (Old Format Packets) 24
4.1.2 Pakete im neuen Format (New Format Packets) 25
4.1.3 Pakettypen 27
4.1.4 Nachrichtentypen 34
4.2 OpenPGP Schlüssel 36
4.3 Radix-64 Konvertierung 36
4.4 Klartext-Signaturen 41

5 Eclipse Platform **42**
5.1 Eclipse Platform Architektur 42
5.2 Eclipse Benutzeroberfläche (Eclipse UI) 46
5.3 Plug-in Architektur 48
5.4 Eclipse Rich Client Platform 50

6 Design und Implementierung des OpenPGP Clients **52**

6.1 Motivation für die Verwendung der Eclipse RCP Platform 52
6.2 Entwicklungsumgebung 53
6.3 Übersicht OpenPGP Client Plug-ins 53
6.4 Analyse der bestehenden OpenPGP-Bibliothek 54
6.5 Neuentwicklung der OpenPGP Bibliothek 57
6.5.1 Verwaltung der Algorithmen 58
6.5.2 Nachrichtenverarbeitung 61
6.5.3 Schlüsselerzeugung 67
6.5.4 Schlüsselspeicherung 67
6.5.5 Schlüsselimport/-export 71
6.5.6 Vertrauensmodelle 73
6.6 OpenPGP Client Benutzeroberfläche (UI) 73
6.6.1 Aufbau des Hauptfensters 73
6.6.2 Individuelle Anpassung der Eclipse-Oberfläche 77
6.6.3 Darstellung der Hauptfunktionen 80
6.6.4 Benutzereinstellungen 83
6.7 Benutzerhandbuch 84
6.8 Internationalisierung 85
6.9 Fehlerprotokollierung 86

7 Zusammenfassung **90**

Literaturverzeichnis **91**

Abbildungsverzeichnis

2.1 Ver- und Entschlüsselung einer Nachricht 5
2.2 Signaturerzeugung und Verifikation eines Dokuments 10

3.1 Beispiel eines Vertrauensbaums 12
3.2 Komponenten einer hierarchischen Public-Key-Infrastruktur 14
3.3 Beispiel für ein einfaches Vertrauensnetz 18
3.4 Beispielgraph mit einem Pfad der Länge 3 20
3.5 Darstellung eines Vertrauensnetzes als probabilistischer Graph 21

4.1 Struktur einer OpenPGP Nachricht 24
4.2 Struktur des Packet Tag 24
4.3 Verschiedene Längentypen im alten Paketformat 25
4.4 Beispiel für eine partielle Kodierung eines Datenpakets 26
4.5 Struktur eines Symmetric-Key Encrypted Session Key Packet 30
4.6 Struktur eines Public Key/Subkey Packet der Version 4 31
4.7 Struktur eines Signature Packet der Version 4 34
4.8 Syntaktische Regeln für die Komposition von OpenPGP Nachrichten . . 35
4.9 Beispiel für eine mehrfach verschachtelte OpenPGP Nachricht 35
4.10 Struktur eines OpenPGP Schlüssels Version 4 37
4.11 Prinzip der Base64-Kodierung 38

5.1 Eclipse Platform Architektur 43
5.2 Eclipse Workbench 47

6.1 Plug-in Struktur der OpenPGP Bibliothek 58
6.2 Vererbungshierarchie von `IAlgorithm` 60
6.3 Beispiel für die Verarbeitung einer OpenPGP Nachricht mit Filtern . . . 62
6.4 Hierarchie der von `FilterInputStream` abgeleiteten Klassen 64
6.5 Vererbungshierarchie von `IMessage` 65
6.6 Hierarchie der Datenbank-Klassen 68
6.7 Vererbungshierarchie von `ITrustModel` 74
6.8 Aufbau des Hauptfensters 76
6.9 Die Werkzeugleiste des OpenPGP Clients 78
6.10 Schlüsselimport über einen Schlüsselserver 83
6.11 Präferenzseiten in den Benutzereinstellungen 84
6.12 Darstellung von Einträgen aus dem Fehlerprotokoll im Error Log View . 88
6.13 Details eines Fehlerprotokoll-Eintrags 89

Tabellenverzeichnis

2.1 Vergleich symmetrische-asymmetrische Kryptographie 8

3.1 Struktur eines X.509 bzw. X.509v3 Zertifikats 13

4.1 OpenPGP Pakettypen 27
4.2 Base64 Umsetzungstabelle 38

Kapitel 1

Einleitung

1991 erschien die erste Version der, von Phil Zimmermann entworfenen kryptographischen Software, *Pretty Good Privacy* (PGP) [Gar96]. Zu den Grundfunktionen von PGP zählten die Verschlüsselung und Signierung von Dateien und E-Mails. PGP bediente sich dabei starker symmetrischer und asymmetrischer kryptographischer Verfahren und stellte insofern ein Novum dar, dass es zum ersten Mal einer breiten Allgemeinheit die Möglichkeit zum Schutz der eigenen (digitalen) Privatsphäre ermöglichte. Um die Interoperabilität anderer Programme mit PGP zu ermöglichen, wurden die von PGP verwendeten Algorithmen und Protokolle im Jahr 1998 in einem offenen Standard der *Internet Engineering Task Force* (IETF) formalisiert. Der Standard wurde im RFC 2440[1] als *OpenPGP Nachrichten Format* veröffentlicht. Das OpenPGP Format basiert auf der PGP Version 5.x und ist heute, neben der weit verbreiteten Alternative *S/MIME*[2] (Secure/Multipurpose Internet Mail Extensions), einer der wichtigsten Standards zum Verschlüsseln und Signieren von E-Mails. Die aktuelle Version der Spezifikation findet sich im RFC 4880 [CDF+07]. Die bekanntesten OpenPGP-Programme sind das heute kommerziell vertriebene PGP[3] und das unter der GNU General Public License (GPL) stehende *GNU Privacy Guard* (GnuPG)[4].

Im Wintersemester 2005/06 wurde am Institut für Kryptographie und Computeralgebra der Technischen Universität Darmstadt im Rahmen eines Praktikums mit der Implementierung einer OpenPGP-Programm-Bibliothek, basierend auf dem im RFC 2440 definierten Format begonnen. Die Bibliothek wurde in der Programmiersprache Java entwickelt und verwendet die *Java Platform, Standard Edition*[5](J2SE).

1.1 Aufgabenstellung

Das Ziel dieser Diplomarbeit ist der Entwurf und die prototypische Erstellung einer kryptographischen Anwendung, im folgenden als OpenPGP-Client bezeichnet, welche die

[1] RFC steht für Request for Comments. Dabei handelt es sich um, von der IETF herausgegebene, technische Dokumente.
[2] Siehe auch http://tools.ietf.org/html/rfc1847 und http://tools.ietf.org/html/rfc2311
[3] Siehe auch www.pgp.com
[4] Siehe auch http://www.gnupg.org
[5] J2SE ist verfügbar auf http://java.sun.com/javase/

komfortable Nutzung der OpenPGP-Bibliothek über eine graphische Benutzeroberfläche (GUI[6]) ermöglicht.

Die Hauptfunktionen der zu erstellenden Anwendung sind:

- Signieren und Verschlüsseln

  Die Anwendung soll sowohl symmetrische, als auch asymmetrische Ver- und Entschlüsselung, sowie das Signieren und Verifizieren von Daten ermöglichen.

- Schlüssel- und Zertifikatsverwaltung

  Die Applikation erlaubt eine einfache Schlüssel- und Zertifikatsverwaltung. Es können neue Schlüssel generiert und bestehende Schlüssel bearbeitet werden. Die Schlüssel werden durch Passwörter vor unbefugtem Zugriff geschützt.

- Internationalisierung (i18n[7])

  Die Anwendung soll wahlweise in deutscher oder englischer Sprache verfügbar sein.

- Fehlerprotokollierung

  Treten Fehler bei der Verwendung der Software auf, sollen diese für eine spätere Analyse in einer Datei gespeichert werden. Dieser Vorgang wird auch als *Logging* bezeichnet.

- Plattformunabhängigkeit

  Die Anwendung soll sowohl unter Linux, als auch unter Windows ausführbar sein.

Die OpenPGP-Bibliothek stellt zum aktuellen Zeitpunkt noch keine vollständige Implementation des OpenPGP-Standards dar. Zudem wird der Standard in unregelmäßigen Abständen aktualisiert, was eine eventuelle Anpassung der Bibliothek nach sich zieht. Schwerpunkt bei der Umsetzung des Clients ist daher eine flexible Software-Architektur, die es auf einfache Art und Weise ermöglicht, neue Versionen der Bibliothek zu integrieren. Dies soll durch den sinnvollen Einsatz von Design Patterns erreicht werden.

Die Implementation des Clients erfolgt in Java und wird als *Eclipse Rich Client Platform*-Anwendung (Eclipse RCP[8]) konzipiert.

1.2 Gliederung der Arbeit

Kapitel 2 erläutert die Sicherheitsziele, die durch die Verwendung von kryptographischer Software erreicht werden sollen, sowie die grundlegenden kryptographischen Verfahren

[6] GUI ist die Abkürzung für den englischen Begriff *Graphical User Interface*.

[7] Der Begriff der Internationalisierung wird oft mit i18n abgekürzt, bezogen auf die Anzahl der 18 ausgelassenen Buchstaben im englischen Wort *internationalization*.

[8] Siehe http://www.eclipse.org/home/categories/rcp.php

wie Verschlüsselung, Hash-Funktionen und Signaturen. Dies soll das Verständnis der später beschriebenen Implementierung des OpenPGP-Clients erleichtern.

In Kapitel 3 wird der Begriff der Public-Key-Infrastruktur (PKI) eingeführt. Danach werden die elementaren Komponenten von hierarchisch organisierten Public-Key-Infrastrukturen dargestellt. Abschließend werden zwei Beispiele für dezentral verwaltete PKI-Lösungen vorgestellt; das klassische, in PGP eingeführte *Netz des Vertrauens (Web of Trust)*, sowie das, an der Universität Bern entwickelte, *probabilistische Vertrauensmodell*.

Kapitel 4 gibt einen Überblick über das *OpenPGP Nachrichten Format* und stellt die wichtigsten Elemente und Strukturen vor.

In Kapitel 5 wird die Architektur der *Eclipse Platform*[9] und die Struktur von *Eclipse RCP*-Anwendungen erläutert.

Kapitel 6 beschreibt die, an der OpenPGP-Bibliothek durchgeführten, notwendigen Modifikationen und Erweiterungen, sowie den Entwurf und die Umsetzung des OpenPGP-Clients.

Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und gibt ein kurzes Fazit.

[9]Siehe http://www.eclipse.org/

Kapitel 2

Kryptographische Grundlagen

Dieses Kapitel stellt nach einer allgemeinen Einleitung in das Thema zunächst die Sicherheitsziele der modernen Kryptographie vor. Im Anschluss folgt eine überblicksartige Darstellung der grundlegenden kryptographischen Konzepte, die für das Verständnis der in Kapitel 5 beschriebenen Implementierung der Anwendung notwendig sind. Die einzelnen Verfahren werden nicht im Detail erläutert. Für weiterführende Informationen zu diesem Thema wird auf die Literatur verwiesen z.B. [Buc03], [Sch06] und [BNS05].

Die Kryptographie im klassischen Sinn war die Lehre von der *Datenverschlüsselung* [Buc03]. Sie hatte primär das Ziel die vertrauliche Kommunikation von Nachrichten bzw. Speicherung von Daten zu gewährleisten. Dies wurde durch die Verwendung von Verschlüsselungsverfahren ermöglicht.

Heutzutage findet die Kryptographie in den verschiedensten Bereichen Anwendung. Unternehmen senden vertrauliche und ggf. wettbewerbsrelevante Informationen, für die sich potentielle Mitbewerber interessieren könnten, über unsichere Medien wie das Internet. Sicherheitsrelevante Software muss vor der Benutzung auf Integrität überprüft werden, um unberechtigte Modifikationen zu verhindern. Beim Abschluss von digitalen Verträgen müssen die Vertragspartner den elektronischen Dokumenten eindeutig zuzuordnen sein. Zudem muss beweisbar sein, dass der Vertrag tatsächlich zustande gekommen ist. Dies sind nur einige Beispiele für die Verwendung von kryptographischen Verfahren, die allerdings zeigen, dass sich die moderne Kryptographie nicht mehr alleine auf die Anforderung nach Vertraulichkeit beschränkt.

[Buc03] definiert folgende Sicherheitsziele:

- Vertraulichkeit

  Vertraulichkeit soll gewährleisten, dass keine unberechtigte Informationsgewinnung über Daten in gespeicherter oder übertragener Form möglich ist. Wie in der Einleitung beschrieben wird dies durch die Verwendung von Verschlüsselungsverfahren sichergestellt.

- Authentizität

  Unter *Authentizität* versteht man die Echtheit und Glaubwürdigkeit von Informationen. Diese kann durch Überprüfung der Identität des Absenders der Informatio-

nen mittels kryptographischer Techniken garantiert werden. Dieser Vorgang wird auch als Authentifikation bezeichnet.

- Integrität

  Die *Integrität* von Daten oder Programmen garantiert, dass diese nicht unberechtigt und unbemerkt manipuliert wurden. Dies läßt sich anhand von kryptographischen Verfahren wie Hashfunktionen überprüfen.

- Nicht-Abstreitbarkeit

  Die *Nicht-Abstreitbarkeit* elektronischer Dokumente macht es Dritten gegenüber beweisbar, dass ein Dokument von einem bestimmten Absender kommt. Dies wird durch die Verwendung von digitalen Signaturen ermöglicht.

2.1 Verschlüsselung

Ein einfaches Beispiel soll zu Beginn die Funktion von Verschlüsselungsverfahren illustrieren. Angenommen zwei Kommunikationspartner Alice und Bob möchten eine Nachricht m vertraulich über ein unsicheres Medium austauschen. Die zu sendende Nachricht wird in der Regel als *Klartext* bezeichnet. Alice verschlüsselt die Nachricht mit einem Verschlüsselungsschlüssel e und einer Verschlüsselungsfunktion $E_e(m)$ und generiert damit den sogenannten *Chiffretext* oder auch *Schlüsseltext* c. Es gilt also $E_e(m) = c$. Alice schickt den Chiffetext nun zum Empfänger Bob. Dieser entschlüsselt die kodierte Nachricht mit einem Entschlüsselungsschlüssel d und der entsprechenden Entschlüsselungsfunktion $D_d(c) = m$, um den ursprünglichen Klartext wieder herzustellen. Dieser Sachverhalt wird in Abbildung 2.1 dargestellt.

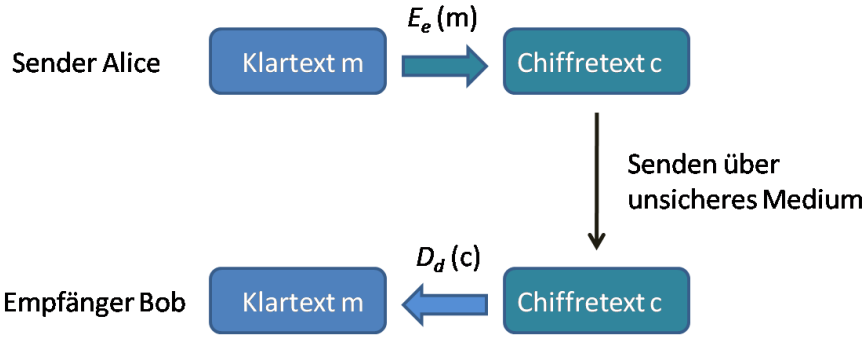

Abbildung 2.1: Ver- und Entschlüsselung einer Nachricht

Die folgende formale Definition von Verschlüsselungsverfahren findet sich in [Buc03]:

Definition 2.1.1 *Ein Verschlüsselungsverfahren oder Kryptosystem ist ein Fünftupel* (P, C, K, E, D) *mit folgenden Eigenschaften:*

1. P *ist eine Menge. Sie heißt Klartextraum. Ihre Elemente heißen Klartexte.*
2. C *ist eine Menge. Sie heißt Chiffretextraum. Ihre Elemente heißen Chiffretexte oder Schlüsseltexte.*
3. K *ist eine Menge. Sie heißt Schlüsselraum. Ihre Elemente heißen Schlüssel.*
4. $E = \{E_k : k \in K\}$ *ist eine Familie von Funktionen* $E_k : P \to C$*. Ihre Elemente heißen Verschlüsselungsfunktionen.*
5. $D = \{D_k : k \in K\}$ *ist eine Familie von Funktionen* $D_k : C \to P$*. Ihre Elemente heißen Entschlüsselungsfunktionen.*
6. *Für jedes* $e \in K$ *gibt es ein* $d \in K$*, so dass für alle* $p \in P$ *die Gleichung* $D_d(E_e(p)) = p$ *gilt.*

"Ein Kryptosystem besteht [also] aus einem Algorithmus [der die Ver- und Entschlüsselungsfunktionen bestimmt] einschließlich aller möglichen Klartexte, Chiffretexte und Schlüssel" [Sch06].

Man unterscheidet im Allgemeinen zwischen *symmetrischen* oder *Private-Key*-Verfahren und *asymmetrischen* oder *Public-Key*-Verfahren. Bei symmetrischen Algorithmen sind Ver- und Entschlüsselungsschlüssel oft identisch bzw. der Dechiffrierschlüssel läßt sich leicht aus dem Chiffrierschlüssel berechnen. Aus diesem Grund muss der verwendete Schlüssel geheimgehalten und vor der Kommunikation über eine sichere Leitung zwischen den Teilnehmern ausgetauscht werden. Für n Teilnehmer müssen dabei $n * \frac{(n-1)}{2}$ Schlüssel geheim ausgetauscht werden [Buc03]. Bei einer großen Anzahl von Teilnehmern wird dieser Ansatz somit zunehmend weniger praktikabel.

Symmetrische Verfahren lassen sich in die Kategorien der *Stromchiffren* und *Blockchiffren* unterteilen. Stromchiffren verarbeiten die Daten bit- oder byteorientiert, während Blockchiffren die Daten in größeren Bit-Blöcken verarbeiten (die Standard-Blockgröße bei vielen Algorithmen beträgt beispielsweise 64 oder 128 Bit).

Es gibt eine Vielzahl von symmetrischen Algorithmen. Zu den bekanntesten Vertretern gehören unter anderen *3DES*[10] (TripleDES), *Blowfish*[11] und *Rijndael*, der im Jahr 2000 vom *National Institute of Standards and Technology* (NIST) zum *Advanced Encryption Standard*[12] (AES) erklärt wurde.

[10] Siehe csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
[11] Siehe http://www.schneier.com/blowfish.html
[12] Siehe http://csrc.nist.gov/archive/aes/index.html

Einen anderen Ansatz verfolgen Public-Key-Verfahren, die unterschiedliche Ver- und Entschlüsselungsschlüssel nutzen. Jeder Teilnehmer besitzt ein eigenes Schlüsselpaar bestehend aus einem *privaten* (private key) und einem *öffentlichen* Schlüssel (public key). Im Gegensatz zu den symmetrischen Verfahren läßt sich der private Schlüssel nicht in angemessener Zeit aus dem öffentlichen Schlüssel berechnen. Der private Schlüssel verläßt in diesem Szenario optimalerweise niemals den eigenen Besitz und wird in der Regel an einem sicheren Ort verwahrt. Der öffentliche Schlüssel hingegen wird anderen Kommunikationspartnern zur Verfügung gestellt. Ein geheimer Schlüsselaustausch vor der Kommunikation ist somit nicht mehr notwendig. Dies vereinfacht die Schlüsselverwaltung erheblich.

Wenn Alice eine Nachricht an Bob senden möchte, muss sie sich zunächst den öffentlichen Schlüssel von Bob besorgen (z.B. von einem Schlüsselserver). Danach verschlüsselt sie die Nachricht mit dem öffentlichen Schlüssel von Bob und sendet ihm die chiffrierte Nachricht. Bob kann nun mit seinem geheimen privaten Schlüssel den Klartext rekonstruieren. Bei dieser Art Kommunikation ist es wichtig, dass die Authentizität des öffentlichen Schlüssels gewährleistet ist. Das heißt es muss sichergestellt werden, dass der öffentliche Schlüssel auch wirklich dem entsprechenden Empfänger (in diesem Fall also Bob) gehört. Kapitel 3 beschäftigt sich näher mit diesem Thema. Ein weiteres Problem von Public-Key-Verfahren ist die Verarbeitungsgeschwindigkeit. Da die Algorithmen mit sehr großen Zahlen arbeiten, ist die Performance im Vergleich zu symmetrischen Verfahren sehr langsam. Sie eignen sich daher nicht für die Anwendung bei großen Datenmengen.

Das bekannteste und meist verwendete Public-Key-Verfahren ist das *RSA*[13]-Verfahren, dessen Sicherheit auf der Primfaktorzerlegung großer Zahlen basiert. Ein weiterer wichtiger Vertreter ist das *ElGamal*-Verfahren[14]. Die Sicherheit von ElGamal beruht auf dem Problem der Berechnung von diskreten Logarithmen.

Ein Vergleich der Eigenschaften von symmetrischen und asymmetrischen Verfahren findet sich in Tabelle 2.1.

Um die Stärken beider Ansätze zu nutzen wird in der Praxis oft eine Kombination aus symmetrischen und asymmetrischen Verfahren verwendet, die als *Hybridverfahren* bezeichnet wird. Dabei wird die eigentliche Nachricht mit einem symmetrischen *Sitzungsschlüssel* (session key) verschlüsselt. Der Sitzungsschlüssel selbst wird mit dem öffentlichen Schlüssel jedes Empfängers asymmetrisch chiffriert und mit der Nachricht versendet. Jeder Empfänger kann dann den Sitzungsschlüssel und damit auch die Nachricht mit seinem privaten Schlüssel entschlüsseln.

[13]Benannt nach den Erfindern Ronald L. Rivest, Adi Shamir und Leonard Adleman. Siehe hierzu http://theory.lcs.mit.edu/ rivest/rsapaper.pdf.

[14]Siehe http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/18/22749/01057074.pdf

Symm. Algorithmen	**vs.**	**Asymm. Algorithmen**
sehr viele	Anzahl	sehr wenige
kann sehr gut sein	Sicherheit	kann sehr gut sein
in der Regel sehr gut	Performance	nicht gut
ja	Vorheriger Schlüssel-austausch notwendig	nein
nein	Möglichkeit der digitalen Signatur	ja
Verschlüsselung	typisches Einsatzgebiet	Signaturen Schlüsselaustausch

Tabelle 2.1: Vergleich symmetrische-asymmetrische Kryptographie. Quelle: [BNS05].

2.2 Hash-Funktionen

Eine Hashfunktion h ist eine Abbildung, die Zeichenketten beliebiger Länge auf Zeichenketten fixer Länge abbildet:

$$h : \Sigma^* \rightarrow \Sigma^n, n \in \mathbb{N}.$$

Hashfunktionen finden in verschiedenen Bereichen Anwendung. Sie werden unter anderem in Datenbanksystemen (DBS) für das schnelle Auffinden von Werten mittels sogenannter *Hashtabellen* oder bei der Fehlerüberprüfung von übermittelten Daten benutzt. Im kryptographischen Sektor werden sie für die Überprüfung der Integrität von Daten eingesetzt und sind ein wichtiger Bestandteil von digitalen Signaturen, die im nächsten Abschnitt behandelt werden.

Kryptographische Hashfunktionen müssen nach [Buc03] allerdings folgende zusätzliche Anforderungen erfüllen:

Sei h eine Hashfunktion und $D = \Sigma^*$ der Definitionsbereich von h.

1. Der Hashwert $h(x)$ muss für alle $x \in D$ effizient berechenbar sein.
2. Es ist praktisch unmöglich für einen gegebenen Hashwert $y \in \Sigma^n$ ein $x \in D$ zu finden mit $h(x) = y$.

Erfüllt h die obigen Eigenschaften bezeichnet man sie auch als *Einwegfunktion*.

Manche Anwendungen erfordern zusätzlich das h *stark kollisionsresistent* ist. Eine *Kollision* von h ist ein Paar $(x, x') \in D^2$ mit $x \neq x'$ und $h(x) = h(x')$. Eine Abbildung h wird als *schwach kollisionsresistent* bezeichnet, wenn es praktisch unmöglich ist, zu gegebenem $x \in D$ ein $x' \in D$ zu finden, so dass das Paar (x, x') eine Kollision von h ist. Die Abbildung h ist *stark kollisionsresistent*, wenn es praktisch unmöglich ist ein beliebiges Paar $(x, x') \in D^2$ zu finden, so dass (x, x') eine Kollision von h ist.

Eines der am meisten genutzten Hashverfahren ist der *Secure Hash Algorithm* (SHA-1), der vom NIST entwickelt wurde und Teil des *Digital Signature Standards*[15] (DSS) ist. SHA-1 bildet Zeichenketten auf 160 Bit Strings ab. Im Jahr 2005 konnten jedoch erfolgreiche Kollisionsangriffe gegen SHA-1 durchgeführt werden, die die Sicherheit des Verfahrens langfristig in Frage stellen[16]. Alternativen sind beispielsweise RIPEMD-160[17] oder die SHA-2[18] Familie (SHA-224, SHA-256, SHA-384 und SHA-512), deren Algorithmen auf SHA-1 basieren, aber Hashwerte größerer Bitlänge erzeugen.

2.3 Digitale Signaturen

Digitale Signaturen sollen die Eigenschaften gewöhnlicher, handschriftlicher Unterschriften auf elektronische Medien übertragen.

[Eck04] beschreibt die folgenden Funktionen einer herkömmlichen Unterschrift:

- Identifikation

  Eine identifizierbare Unterschrift "gibt Auskunft über die Person des Unterzeichners" [Eck04].

- Echtheit

  Die Echtheit einer Unterschrift "bezeugt, dass das Dokument dem Austeller vorgelegen hat und von ihm anerkannt wurde" [Eck04].

- Abschluss

  Der Abschluss einer Unterschrift "erklärt den Text für inhaltlich richtig und vollständig" [Eck04].

- Warnung

  Die Warnung einer Unterschrift besagt, dass durch "die Notwendigkeit, dass eine Unterschrift geleistet werden muss [...] dem Verfasser die rechtliche Bedeutung des Dokuments aufgezeigt [wird]" [Eck04].

Ziel von digitalen Signaturen ist es daher die Authentizität, Integrität und Nicht-Abstreitbarkeit von elektronischen Dokumenten zu garantieren. Wie in der Einleitung bereits erläutert, ist dies zum Beispiel für den Abschluss von digitalen Verträgen von großer Bedeutung.

Signaturen nutzen Public-Key-Kryptographie, das heißt um eine Signatur zu erstellen und zu verifizieren benötigt man, wie in Abschnitt 2.1 beschrieben, ein Schlüsselpaar

[15] Siehe http://www.itl.nist.gov/fipspubs/fip186.htm
[16] Siehe auch http://www.schneier.com/blog/archives/2005/02/sha1_broken.html
[17] Siehe http://homes.esat.kuleuven.be/ cosicart/pdf/AB-9601/AB-9601.pdf
[18] Siehe http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

bestehend aus einem privaten und einem öffentlichen Schlüssel. Die Funktion der beiden Schlüssel wird jedoch vertauscht. Angenommen Alice möchte ein Dokument d signieren. Zunächst wird der Hashwert des Dokuments $h(d)$ berechnet. Der resultierende Wert wird mit dem privatem Schlüssel von Alice verschlüsselt und als Signatur s an das Dokument angehängt. Will nun eine dritte Partei die Signatur des Dokuments überprüfen, wird erneut der Hashwert des vorliegenden Dokuments $h(d)'$ berechnet. Mit dem öffentlichen Schlüssel von Alice wird der ursprüngliche Wert $h(d)$ entschlüsselt und es läßt sich verifizieren, ob die beiden Hashwerte übereinstimmen, also $h(d) = h(d)'$ gilt. Ist dies der Fall, ist die Signatur gültig. Dieser Prozess ist in Abbildung 2.2 dargestellt.

Zu den meist verwendeten Signaturalgorithmen zählen die *RSA-Signatur*, sowie der *Digital Signature Algorithm*[19] (DSA). DSA wurde 1991 vom NIST vorgeschlagen und später in den DSS integriert. Im Gegensatz zur RSA-Signatur handelt es sich bei DSA um ein reines Signaturverfahren, es existiert kein verwandtes Verschlüsselungsverfahren. Demzufolge kann DSA nicht für Verschlüsselung benutzt werden.

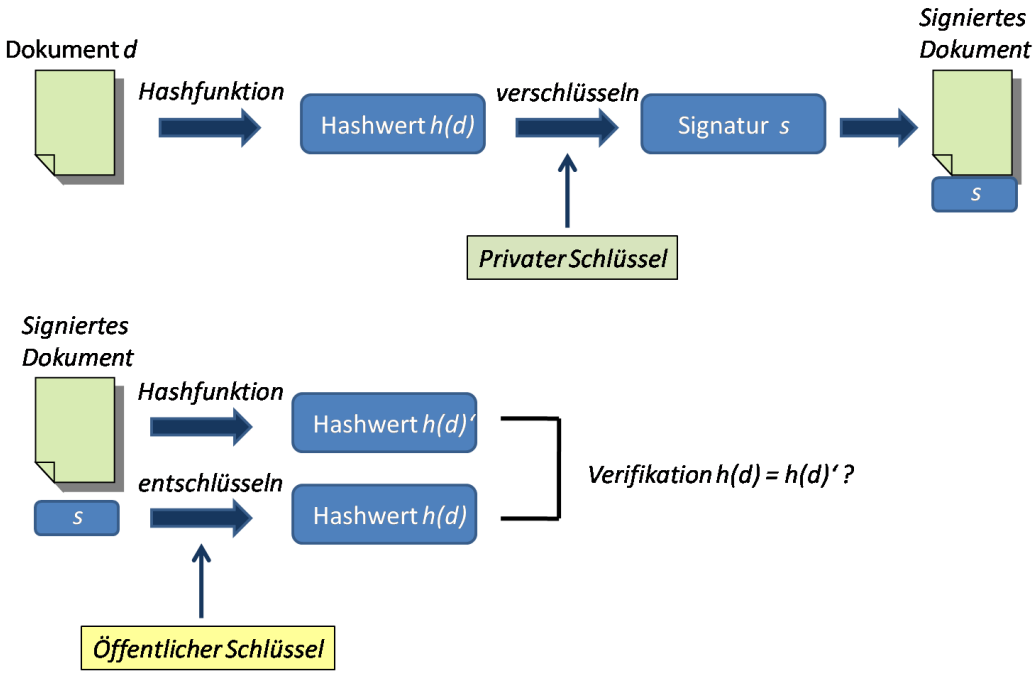

Abbildung 2.2: Signaturerzeugung und Verifikation eines Dokuments

[19]Siehe http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

Kapitel 3

Public-Key-Infrastrukturen

Dieses Kapitel beschreibt zunächst den Begriff der *Public-Key-Infrastruktur* (PKI) sowie die grundlegenden Komponenten einer hierarchisch verwalteten PKI. Im folgenden werden zwei Beispiele für dezentrale Public-Key-Infrastrukturen vorgestellt, das *Netz des Vertrauens (Web of Trust)*, sowie das *probabilistische Vertrauensmodell*.

Wie in Kapitel 2 bereits erläutert wurde, ist eines der Probleme der Public-Key-Kryptographie der Nachweis der Authentizität von öffentlichen Schlüsseln, das heißt es muss sichergestellt werden, dass ein öffentlicher Schlüssel tatsächlich dem vermeintlichen Besitzer gehört. Andernfalls könnte ein Angreifer einen Schlüssel unter einer falschen Identität herausgeben, um sich so unberechtigt Daten anzueignen. Dieses Problem wird auch als *Schlüsselvalidierungsproblem (key validation problem)* bezeichnet [JWH06]. Um es zu lösen verwendet man *Public-Key-Infrastrukturen* (PKI).

Eine Public-Key-Infrastruktur ist ein komplexes System, dass über definierte Dienste, die sichere Verwaltung und Verwendung von asymmetrischen Schlüsseln ermöglichen soll. Zu diesen Diensten zählen die Erzeugung, Verwaltung und der Rückruf von digitalen Zertifikaten. Ein *digitales Zertifikat* bindet dabei, vergleichbar einem Personalausweis, eine Person oder eine Institution eindeutig an einen öffentlichen Schlüssel. Dies wird durch die Verwendung von Signaturen erreicht. Es gibt verschiedene Ansätze eine Public-Key-Infrastruktur zu implementieren. Dazu zählen hierarchisch und dezentral organisierte Infrastrukturen, die im folgenden beschrieben werden.

3.1 Hierarchisch organisierte Public-Key-Infrastruktur

In einer *hierarchisch organisierten* PKI werden die Zertifikate von einer zentralen unabhängigen Instanz, der *Zertifizierungsstelle* (Certification Authority, CA) herausgegeben. Eine CA kann auch Zertifikate für untergeordnete CAs erstellen. Daraus ergibt sich eine Hierarchie, die in [Buc05] als *Vertrauensbaum* (trust tree) bezeichnet wird. Am unteren Ende dieses Baums stehen die Endanwender, wie Personen oder Firmen. Dies ist in Abbildung 3.1 dargestellt. Die Sicherheit einer solchen PKI basiert dabei auf dem uneingeschränkten Vertrauen in die oberste Zertifizierungstelle, die *Wurzelinstanz* (Root-CA). Sie wird auch als *Vertrauensanker* bezeichnet (trust anchor). Wenn Bob in dem dargestellen Beispiel das Zertifikat von Alice überprüfen möchte, verifiziert

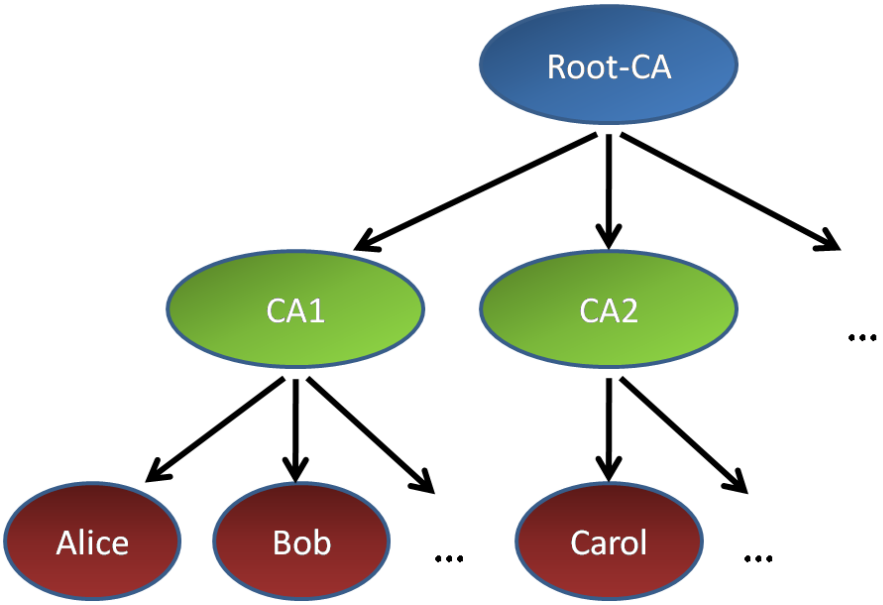

Abbildung 3.1: Beispiel eines Vertrauensbaums

er zunächst das Zertifikat von *CA1* mit dem öffentlichen Schlüssel der Wurzelinstanz *Root-CA*. Ist das Zertifikat gültig, prüft er das Zertifikat von Alice mit dem öffentlichen Schlüssel von *CA1*. Ist auch diese Verifikation positiv, gilt der Schlüssel von Alice als authentisch[20].

Wenn eine Person oder Institution ein Zertifikat beantragen möchte, muss sie sich zunächst einer *Registrierungsstelle* (Registration Authority, RA) gegenüber identifizieren und mit Namen und weiteren persönlichen Informationen registrieren[21]. Die Identifikation der angegebenen Daten kann z.B. anhand der persönlichen Vorlage des Lichtbildausweises oder Führerscheins, aber auch über elektronische Medien wie das Internet erfolgen. Die Registrierungsstelle überprüft die angegebenen Daten. Im Falle einer positiven Verifikation wird dem Antragsteller ein eindeutiger Benutzername zugewiesen und der Zertifikatsantrag mit allen notwendigen Informationen an die Zertifizierungsstelle weitergeleitet. Eine Voraussetzung für einen Zertifikatsantrag ist natürlich ein asymmetrisches Schlüsselpaar. Dieses kann entweder vom Antragsteller selbst erzeugt werden (in diesem Fall muss die RA prüfen, dass der Antragsteller tatsächlich im Besitz des privaten Schlüssels ist) oder von der Registrierungsstelle (in diesem Fall muss sichergestellt werden, dass ausschließlich der Antragsteller den generierten privaten Schlüssel erhält). Die tatsächliche Erstellung und Herausgabe des Zertifikats wird von der Zertifizierungsstelle durchgeführt. Sie signiert das Zertifikat mit ihrem privaten Schlüssel. Die Signatur

[20] Den Pfad von einer Entität zu der vertrauten Zertifizierungsstelle nennt man *Zertifizierungspfad* (certification path). Aufgrund der Eigenschaften eines Baums existiert genau ein solcher Pfad [Buc05].

[21] Eine Zertifizierungsstelle kann dabei durchaus mehrere Registrierungsstellen beschäftigen. Die Zusammenfassung von Registrierungs- und Zertifizierungsstelle bezeichnet man oft auch als *Trust-Center* [Eck04].

kann dann von Dritten jederzeit mit dem öffentlichen Schlüssel der Zertifizierungsstelle überprüft werden.

Ein erstelltes Zertifikat enthält mindestens die folgenden Informationen [Buc05]:

- Benutzername des Antragstellers
- Öffentliche Schlüssel des Antragstellers
- Für die öffentlichen Schlüssel verwendete Algorithmen
- Zertifikats-Seriennummer
- Gültigkeitszeitraum des Zertifikats
- Name des Zertifikatsausstellers
- Informationen über Einschränkungen der Zertifikatsverwendung

Das zur Zeit wichtigste Format für digitale Zertifikate ist der ITU-T-Standard *X.509*[22]. Tabelle 3.1 zeigt die Struktur eines Zertifikats in diesem Format.

Inhalt	**Erläuterung**
Versionsnummer	beschreibt verwendetes Zertifikatsformat
Seriennummer	eindeutiger Identifikator
Signatur	verwendete Algorithmen und Parameter
Zertifikataussteller	Name der ausstellenden Instanz
Gültigkeitsdauer	Angabe eines Zeitintervalls
Benutzername	eindeutiger Name des Benutzers
Schlüsselinformationen	Schlüssel des Benutzers und Algorithmen
eindeutiger Identifikator	in Version v2, v3
Erweiterungen	in Version v2, v3

Tabelle 3.1: Struktur eines X.509 bzw. X.509v3 Zertifikats. Quelle: [Eck04].

Die Zertifizierungstelle veröffentlicht neue Zertifikate in einem *Verzeichnis* (Directory). Benutzer der PKI können dort nach Zertifikaten suchen, wenn sie mit einer bestimmten Entität kommunizieren wollen. Zudem enthält das Verzeichnis Informationen über Zertifikate die vor Ablauf ihrer Gültigkeitsdauer widerrufen wurden, die sogenannte *Zertifikatssperrliste* (Certificate Revocation List, CRL). Mögliche Gründe für einen frühzeitige Sperrung sind z.B. der Verlust oder die Kompromittierung des privaten Schlüssels.

Jeder CRL-Eintrag enthält dabei laut [Buc03]:

- Seriennummer des Zertifikats

[22]Siehe auch http://tools.ietf.org/html/rfc3280

- Zeitpunkt des Widerrufs
- Ggf. weitere Informationen wie z.B. Gründe für den Widerruf

Die einzelnen Einträge werden von der Zertifizierungsstelle signiert. Der Verzeichnisdienst verwendet für den Zugriff oft das *Lightweight Directory Access Protocol*[23] (LDAP), in selteneren Fällen den ITU-T Standard *X.500*[24]. Abbildung 3.2 zeigt eine Übersicht der beschriebenen PKI-Komponenten.

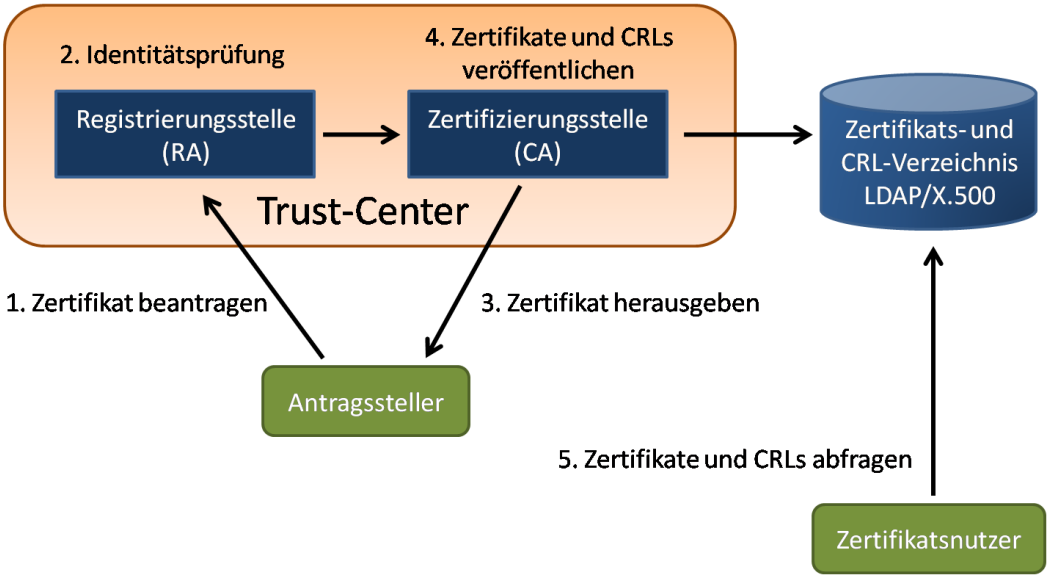

Abbildung 3.2: Komponenten einer hierarchischen Public-Key-Infrastruktur

3.2 Netz des Vertrauens (Web of Trust)

Einen komplett anderen Ansatz verfolgen dezentrale, verteilte Public-Key-Infrastrukturen, deren bekanntester Vertreter das *Netz des Vertrauens* (*Web of Trust*) darstellt. Es wird von PGP, GnuPG und anderen OpenPGP-Lösungen verwendet, daher wird es auch als *PGP Vertrauensmodell* (PGP Trust Model) [AR96] bezeichnet. Im Unterschied zu den zentral verwalteten Infrastrukturen existieren in diesem Konzept keine designierten Zertifizierungsinstanzen. Stattdessen kann jeder Teilnehmer des Netzes Zertifikate für andere Teilnehmer ausstellen. Das Vertrauensnetz wird dabei durch die zertifizierten Schlüssel aufgebaut, wobei die Zertifikate als Verbindungen zwischen den Schlüsseln fungieren [AR96]. Das Konzept des Web of Trust wird auch in der zu erstellenden Anwendung eingesetzt. Die konkrete Implementation wird in Kapitel 6 beschrieben.

[23] Siehe auch http://tools.ietf.org/html/rfc4510
[24] Siehe auch http://www.itu.int/rec/T-REC-X.500/en

Im Folgenden wird die Umsetzung des Web of Trust in PGP beschrieben. Dazu müssen jedoch zunächst einige PGP-Grundbegriffe eingeführt werden. Eine ausführliche Beschreibung des PGP-Formats findet sich in Kapitel 4.

Jeder PGP-Nutzer besitzt ein oder mehrere asymmetrische *Schlüsselpaare* (key pairs), bestehend aus jeweils einem privaten und öffentlichen Schlüssel[25]. PGP speichert diese Schlüsselpaare intern in zwei *Schlüsselbunden* (key rings). Die eigenen und von anderen Teilnehmern importierten, öffentlichen Schlüssel werden in einem *öffentlichen Schlüsselbund* (public key ring) gespeichert, die privaten Schlüssel in einem *geheimen Schlüsselbund* (secret key ring). Der Zugriff auf den geheimen Schlüsselbund wird durch den Einsatz von Passphrasen und symmetrischer Verschlüsselung beschränkt. Schlüssel werden über eine 64-Bit Zahl, die *Schlüsselidentität* (key id) oder über einen *Fingerabdruck* (fingerprint) identifiziert. Der Fingerabdruck ist ein mit SHA-1 berechneter 160-Bit Hashwert über den öffentlichen Schlüssel[26]. Der Besitzer eines Schlüsselpaars wird über eine oder mehrere *Benutzeridentitäten* (user id) beschrieben. Für gewöhnlich besteht eine Benutzeridentität aus einem Namen und einer E-Mail-Adresse in der Form `Name <vor.nachname@domain.tld>`. Es muss jedoch beachtet werden, dass sowohl die Schlüsselidentitäten, als auch die Benutzeridentitäten global nicht eindeutig sind.

Ein Zertifikat ist in PGP eine Signatur, die eine Benutzeridentität an einen öffentlichen Schlüssel bindet. Das erstellte Zertifikat wird an den Schlüssel angehängt. Der Unterzeichner äußert mit seiner Signatur, dass er glaubt, dass der Schlüssel zu der angegebenen Identität gehört. Wird ein öffentlicher Schlüssel mit dem zugehörigen privaten Schlüssel signiert, spricht man auch von einer *Selbstsignatur* (self signing). Erstellt man auf diese Weise ein Zertifikat, nennt man es entsprechend *selbstsigniertes Zertifikat* (self-signed certificate)[27].

Die Validierung eines öffentlichen Schlüssel kann in PGP auf zweierlei Art durchgeführt werden, über eine direkte, persönliche Validierung und über das Netz des Vertrauens.

3.2.1 Direkte Validierung

Angenommen Alice kennt Bob und möchte den öffentlichen Schlüssel von Bob in ihren Schlüsselbund importieren (z.B. von einem Schlüsselserver oder aus einer Datei). Zunächst muss sie die Authentizität des Schlüssels prüfen. Sie vergleicht dazu den Fingerabdruck des importierten Schlüssels mit dem Fingerabdruck von Bobs Schlüssel. Dies kann z.B. per E-Mail, Telefon oder persönlich erfolgen. Wenn sie davon überzeugt ist, dass der Schlüssel tatsächlich Bob gehört, signiert sie ihn mit ihrem eigenen privaten

[25] Der Begriff des Schlüsselpaars ist in PGP mehrdeutig. Tatsächlich kann ein Schlüsselpaar aus mehreren einzelnen Schlüsselpaaren bestehen. Es gibt dann ein *primäres Schlüsselpaar* (primary key) und ein oder mehrere *untergeordnete Schlüsselpaare* (sub keys), die zu einem Schlüsselpaar zusammengefasst werden.

[26] Siehe auch [CDF+07] Sektion 12.2.

[27] Bei der Erstellung eines neuen Schlüsselpaars in PGP werden die zugehörigen Benutzeridentitäten samt Nutzerpräferenzen mit dem eigenen privaten Schlüssel an den öffentlichen Schlüssel gebunden.

Schlüssel. In der Realität kennt man allerdings oft nur einen kleinen Kreis der Kommunikationspartner persönlich. Zudem ist diese Form der Validierung bei einer großen Anzahl von Teilnehmern unpraktisch [Ash99].

3.2.2 Validierung über das Netz des Vertrauens

Im Netz des Vertrauens wird die "[...] Validierung der öffentlichen Schlüssel an vertrauensvolle Teilnehmer delegiert" [Ash99]. Diese werden *Trusted Introducers* genannt. Ein einfaches Beispiel soll dies illustrieren. Angenommen Alice hat Bobs Schlüssel erfolgreich geprüft und signiert. Carl möchte nun mit Bob kommunizieren. Zu diesem Zweck sendet Bob seinen signierten Schlüssel an Carl. Da Carl Alice und somit auch der Signatur von Alice vertraut, hält er Bobs Schlüssel für gültig. Die Authentizität des Schlüssels wird also aus dem Vertrauen zu Alice abgeleitet. PGP unterscheidet daher zwischen dem *Vertrauen in einen Schlüsselbesitzer* (trustworthiness of introducer, oft auch als *owner trust* bezeichnet) und dem *Vertrauen in die Authentizität eines Schlüssels* (trustworthiness of public-key certificate).

3.2.2.1 Vertrauen in einen Schlüsselbesitzer

Dieser Wert gibt an, inwieweit man dem Besitzer eines Schlüssels vertraut gewissenhaft andere Schlüssel zu validieren, bevor er sie selbst zertifiziert. Diese Information wird als sensitiv betrachtet und muss vom PGP-Anwender manuell erfasst werden. Sie wird nicht mit dem Schlüssel selbst, sondern in einer separaten *Vertrauensdatenbank* (trust database) gespeichert.

PGP unterscheidet vier verschiedene Möglichkeiten:

- Unbekanntes Vertrauen (unknown trust)

  Es ist nichts über die Fähigkeit des Schlüsselbesitzers bekannt, andere Schlüssel zu signieren. Dies ist die initiale Einstellung für Schlüssel im öffentlichen Schlüsselbund, die nicht vom Besitzer des Schlüsselbunds erzeugt wurden.

- Kein Vertrauen (no trust)

  Dem Schlüsselbesitzer wird kein Vertrauen entgegengebracht die Schlüssel von Dritten zu signieren.

- Kaum Vertrauen (marginal trust)

  Dem Schlüsselbesitzer wird ein gewisses Maß an Vertrauen entgegengebracht die Schlüssel von Dritten gewissenhaft zu signieren.

- Vollständiges Vertrauen (full trust)

  Dem Schlüsselbesitzer wird vollständig vertraut die Schlüssel von Dritten gewissenhaft zu signieren.

Es existiert noch eine fünfte Vertrauensstufe, das *ultimative oder implizite Vertrauen* (ultimate trust). Dieser Wert wird automatisch vergeben, wenn der zugehörige Schlüssel vom Schlüsselbundeigner erzeugt wurde oder er im Besitz des privaten Schlüssels ist. Diese Schlüssel sind automatisch gültig.

3.2.2.2 Vertrauen in die Authentizität eines Schlüssels

Die Authentizität eines Schlüssels kann, basierend auf den vergebenen Vertrauenswerten für die Schlüsselbesitzer, wie folgt geprüft werden:

Ein Schlüssel s ist nach [AR96] *vollständig gültig* (completely valid), wenn

1. der öffentliche Schlüssel dem Schlüsselbundbesitzer gehört.
2. der öffentliche Schlüssel von mindestens C Teilnehmern signiert wurde, denen vollständig vertraut wird und die selbst im Besitz eines vollständig gültigen Schlüssels sind.
3. der öffentliche Schlüssel von mindestens M Teilnehmern signiert wurde, denen kaum vertraut wird und die selbst im Besitz eines vollständig gültigen Schlüssels sind.

Die Werte C und M stehen für `COMPLETES_NEEDED` und `MARGINALS_NEEDED`. Man nennt sie auch *Skepsis Parameter* (skeptism parameters)[28] Sie "[...] reflektieren die eigenen [Sicherheits-]Regeln des Benutzers hinsichtlich seines [...] Vertrauens in PGP-Signaturen" [AR96].

Formal wird die Berechnung der Schlüsselzulässigkeit (key legitimacy) L in [JWH06] wie folgt dargestellt:

$$L = \frac{c}{C} + \frac{m}{M}$$

Der Wert c entspricht dabei der Anzahl der Teilnehmer mit vollständigem Vertrauen und vollständig gültigem Schlüssel, m entspricht der Anzahl der Teilnehmer mit vollständig gültigem Schlüssel denen kaum Vertrauen ausgesprochen wird, C und M entsprechen den Skepsis Parametern wie oben dargestellt. Dann ist ein Schlüssel *vollständig gültig*, wenn $L \geq 1$, *teilweise gültig* (marginally trusted), wenn $0 < L < 1$ und ungültig für $L = 0$. PGP verwendet standardmäßig die Werte $C = 1$ und $M = 2$, GnuPG hingegen $C = 1$ und $M = 3$.

Abbildung 3.3 zeigt ein einfaches Beispiel für ein Vertrauensnetz mit den berechneten Schlüsselwerten für $M = 2$ und $C = 1$. Ein Pfeil zwischen zwei Teilnehmern A und B bedeutet, dass A den öffentlichen Schlüssel von B signiert hat. Das Vertrauen von Alice in die einzelnen Teilnehmer ist in Klammern aufgeführt.

[28] GnuPG verwendet einen zusätzlichen Parameter, den *maximalen Zertifizierungspfad* (maximum certification path). Ein zu validierender Schlüssel s darf dabei im Web of Trust nicht weiter als die angegebene Pfadlänge von einem eigenen Schlüssel entfernt sein [Ash99]. Der Standardwert ist 5.

Da Alice Besitzerin des Schlüsselbunds ist, ist ihr Schlüssel automatisch vollständig gültig und da sie sowohl Blakes als auch Carols Schlüssel signiert hat, sind diese beiden Schlüssel auch vollständig gültig. Chloes Schlüssel ist ebenfalls vollständig gültig, da zwei Zertifikate von Teilnehmern denen kaum Vertrauen entgegengebracht wird, ausreichend sind, um einen Schlüssel erfolgreich zu validieren. Francis Schlüssel ist nur teilweise gültig, da das Vertrauen in Chloe unbekannt ist und Alice Carol kaum vertraut.

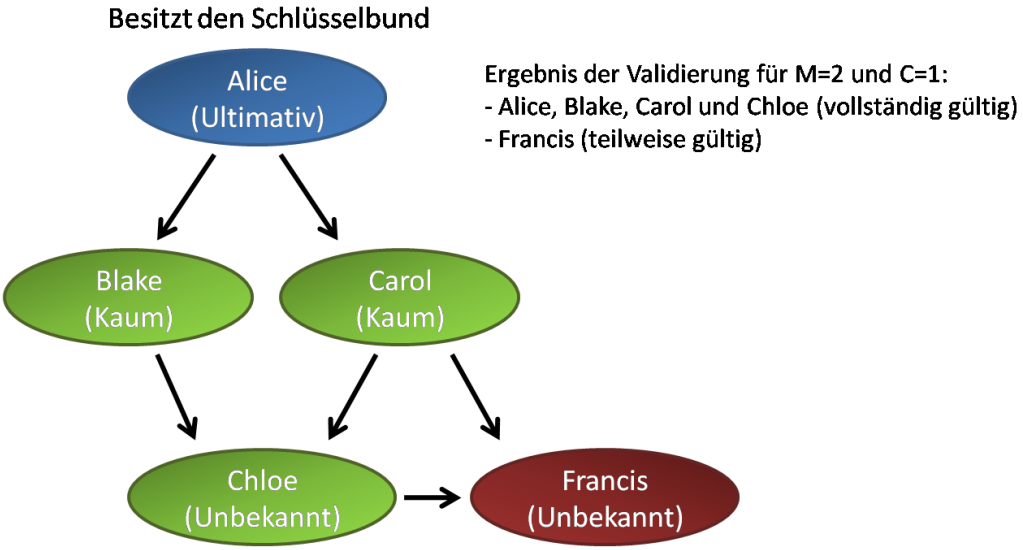

Abbildung 3.3: Beispiel für ein einfaches Vertrauensnetz

3.3 Probabilistisches Vertrauensmodell

In [JWH06] werden einige Aspekte des vorgestellten Netz des Vertrauens kritisiert. Dazu zählen unter anderem:

- Vertrauen in die Schlüsselbesitzer

  PGP unterscheidet nur zwischen vier, eher vage formulierten Vertrauensstufen. Dies ist nach Meinung der Autoren von [JWH06] ggf. nicht ausreichend, um komplexe Vertrauensverhältnisse zu beschreiben.

- Vertrauen in die Schlüsselgültigkeit

  Gleichermaßen ordnet PGP die Gültigkeit eines Schlüssels in nur drei Stufen ein. Diese Kategorisierung ist nach [JWH06] zu unflexibel, um z.B. zu verdeutlichen, dass einem vollständig gültigen Schlüssel eventuell *mehr* Vertrauen entgegengebracht wird, als einem anderen vollständig gültigem Schlüssel.

  Als einfaches Beispiel werden zwei Schlüssel betrachtet, die ausschließlich von Teilnehmern mit *marginal trust* signiert wurden. Die Anzahl der Signaturen ist unter-

schiedlich, liegt in beiden Fällen aber über dem Wert `MARGINALS_NEEDED`. Demzufolge werden beide Schlüssel als vollständig gültig validiert, obwohl "[...] mehr positive Hinweise für eine [stärkere] Gültigkeit des Schlüssels mit der größeren Anzahl an Zertifikaten [existieren]." [JWH06].

- Unsichtbare Abhängigkeiten

  PGP erlaubt seinen Benutzern den Besitz von mehreren Schlüsselpaaren. Demzufolge kann es vorkommen, dass ein Schlüssel zwar die ausreichende Anzahl an notwendigen Signaturen für eine positive Verifikation der vollständigen Gültigkeit besitzt, diese Zertifikate aber mit mehreren Schlüssel von einer einzigen Person ausgestellt wurden. Dies wird in [JWH06] als *unsichtbare Abhängigkeit* (invisible dependency) bezeichnet.

[JWH06] proklamieren daher ein eigenes Modell, das *probabilistische Vertrauensmodell* (probabilistic trust model), welches einen Teil der beschriebenen Probleme des Netz des Vertrauens beheben soll. Das Vertrauensnetz wird in diesem Modell als ein Netzwerk, repräsentiert durch einen probabilistischen Graphen, dargestellt. Im folgenden werden daher zunächst einige grundlegende Begriffe aus der Graphen- und Netzwerkverläßlichkeitstheorie eingeführt.

Ein Graph wird in [Die00] wie folgt definiert:

Definition 3.3.1 *Ein Graph ist ein Paar G=(V,E) disjunkter Mengen mit $E \subseteq [V^2]$, die Elemente von E sind also 2-elementige Teilmengen von V. Die Elemente von V nennt man die Ecken oder Knoten des Graphen G, die Elemente von E seine Kanten.*

Unter einem Pfad versteht man nach [CLRS04]:

Definition 3.3.2 *Ein Pfad der Länge k von einem Knoten u zu einem Knoten u' in einem Graphen G=(V,E) ist eine Folge $\langle v_0, v_1, v_2, ..., v_k \rangle$ von Knoten, so dass $u = v_0, u' = v_k$ und $(v_{i-1}, v_i) \in E$ für $i = 1, 2, ..., k$. Die Länge des Pfades ist die Anzahl der Kanten im Pfad.*

Abbildung 3.4 zeigt einen einfachen Graphen mit dem Pfad $\langle v_0, v_2, v_3, v_4 \rangle$ der Länge 3. In einem *probabilistischen Graph* wird nun jedem Knoten $v \in V$ ein Wert $P(v)$ zwischen 0 und 1 zugeordnet, der die Wahrscheinlichkeit für einen Ausfall des Knotens angibt[29]. Die Kanten gelten in diesem Graph als ausfallsicher. Ein einfaches Beispiel für ein Netzwerk, welches durch solch einen Graphen dargestellt werden kann, ist die Modellierung einer elektronischen Schaltung. Die Knoten des Graphen sind die Bauelemente der Schaltung, die Kanten stellen die Leitungen zwischen den Elementen dar. Die den Knoten zugeordneten Werte, könnten in diesem Fall die Wahrscheinlichkeit für einen Ausfall des Bauelements darstellen. Unter der *Verläßlichkeit eines Netzwerks* (network reliability) versteht man nun nach [Col87] "die Fähigkeit eines Netzwerks[...], eine

[29] Der Wert 0 bedeutet in diesem Fall, dass der Knoten auf keinen Fall ausfällt. Ein Wert von 1 gibt an, dass der Knoten mit absoluter Sicherheit ausfällt.

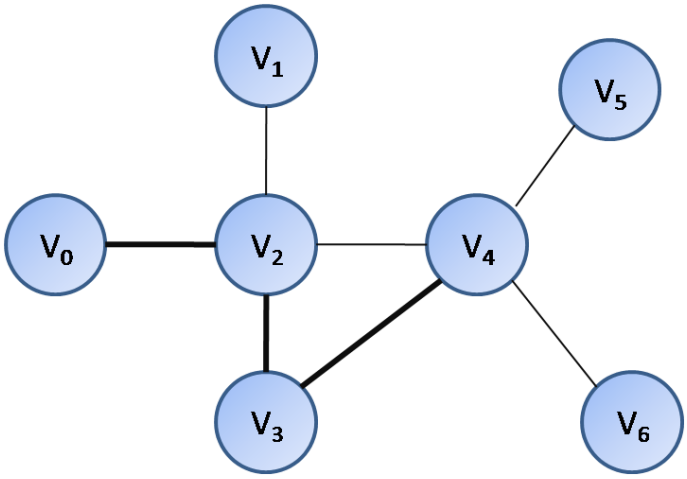

Abbildung 3.4: Beispielgraph mit einem Pfad der Länge 3

gewünschte Operation auszuführen". Auf das obige Beispiel bezogen, könnte man die Frage stellen, mit welcher Wahrscheinlichkeit, basierend auf den vergebenen Werten für die einzelnen Knoten, die Gesamtschaltung immer noch funktioniert.

Eine Ausprägung der Netzwerk-Verläßlichkeit ist die *2-Terminal-Verläßlichkeit* (2-Terminal-Reliability)[30]. Diese wird in [Col87] beschrieben:

Definition 3.3.3 *Für einen probabilistischen Graphen G und festgelegte Knoten s,t, wird die 2-Terminal-Verläßlichkeit, bezeichnet mit $Rel_2(G)$, als die Wahrscheinlichkeit definiert, dass in G wenigstens ein Pfad von s nach t existiert.*

Im probabilistischen Modell wird nun der Prozess der Schlüsselvalidierung in Form eines 2-Terminal-Verläßlichkeits-Problems formuliert. Die Knoten des Netzwerks repräsentieren die Teilnehmer des Vertrauensnetzes, eine Kante zwischen zwei Knoten A und B bedeutet, wie im Netz des Vertrauens auch, dass Teilnehmer A den öffentlichen Schlüssel von B signiert hat. Die den Knoten zugeordneten Wahrscheinlichkeiten entsprechen dem, im Abschnitt 3.2.2.1 beschriebenem Vertrauen in die einzelnen Schlüsselbesitzer. Die Vertrauenswerte können also in diesem Modell flexibel zwischen 0 und 1 vergeben werden. Dies ist einer der Vorteile, die die Entwickler des Modells im Vergleich zum PGP-Modell hervorheben. Ein Wert von 0 bedeutet kein Vertrauen in den Schlüsselbesitzer, ein Wert von 1 steht für vollständiges Vertrauen in den Schlüsselbesitzer[31]. Abbildung 3.5 zeigt ein einfaches Beispiel für ein solches Vertrauensnetz. Analog zu [JWH06] werden in der Abbildung die PGP Vertrauensstufen durch die folgenden Wahrscheinlichkeiten abgebil-

[30] Weitere in [Col87] beschriebene Probleme sind z.B. das *k-Terminal-* und das *All-Terminal-Reliability-Problem*, bei denen es um die Wahrscheinlichkeit der Konnektivität zwischen k, beziehungsweise allen Knoten eines Netzwerks geht.

[31] Die Wahrscheinlichkeiten für die einzelnen Knoten werden im probabilistischen Modell, im Vergleich zum vorgestellten probabilistischen Graphen, in negierter Form angegeben.

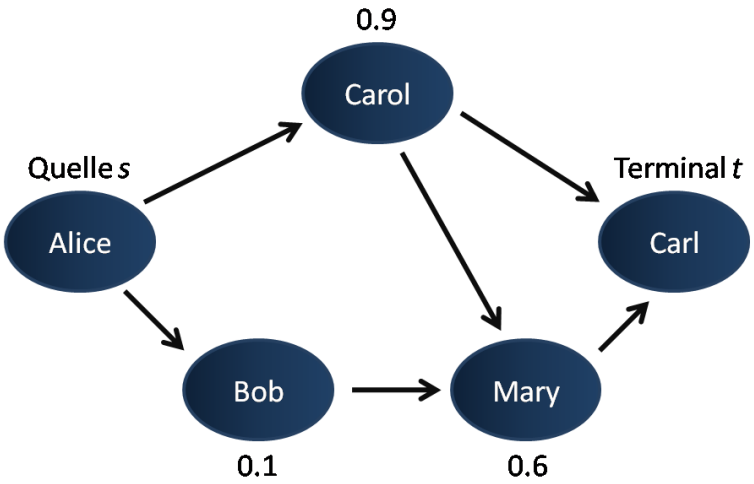

Abbildung 3.5: Darstellung eines Vertrauensnetzes als probabilistischer Graph

det[32]:

- Vollständiges Vertrauen: 0.9
- Kaum Vertrauen: 0.5 - 0.6
- Kein Vertrauen: 0.1

Angenommen Alice möchte Carls Schlüssel validieren. Im probabilistischen Modell wird nun die 2-Terminal-Verläßlichkeit als Maß für die Gültigkeit von Carls Schlüssel herangezogen. Die berechnete Wahrscheinlichkeit wird mit einem von Alice (im Allgemeinem vom Besitzer des Schlüsselrings) vergebenen *Gültigkeits Schwellenwert* (validity threshold) $\lambda \in [0, 1]$ verglichen. Liegt die Wahrscheinlichkeit oberhalb dieses Schwellenwerts, wird der Schlüssel als gültig betrachtet, andernfalls als ungültig. Existiert im Netzwerk kein Pfad zwischen Alice und Carl, kann keine Aussage über die Gültigkeit von Carls Schlüssel getroffen werden (da die Netzwerkverläßlichkeit in diesem Fall nicht berechnet werden kann) [JWH06].

Die Berechnung der 2-Terminal-Verläßlichkeit kann mit Hilfe von exakten oder Näherungsverfahren erfolgen. Wie in [Col87] dargestellt, haben die exakten Verfahren jedoch den Nachteil, dass sie deutlich ineffizienter arbeiten als Approximationsverfahren, letztere aber natürlich nur einen Näherungswert liefern. Eine Beschreibung der konkreten Verfahren geht jedoch über den Rahmen dieser Arbeit hinaus. Hierfür wird auf die Literatur wie z.B. [Col87] oder [JWH06] verwiesen.

[32] Die Unterscheidung zwischen keinem und unbekanntem Vertrauen in einen Schlüsselbesitzer macht für die in Abschnitt 3.2.2.1 vorgestellte Schlüsselvalidierung über das Netz des Vertrauens keinen Unterschied, da dort nur Schlüssel mit kaum und vollständigem Vertrauen mit einbezogen werden. Daher führen die Autoren von [JWH06] die Stufe unbekanntes Vertrauen nicht mit auf.

Kapitel 4

OpenPGP Nachrichten Format

Wie in der Einleitung zu dieser Arbeit bereits erwähnt, stellt OpenPGP neben S/MIME einen der wichtigsten Standards im Bereich der Verschlüsselung und dem Signieren von E-Mails dar. OpenPGP baut auf der von Phil Zimmermann im Jahr 1991 entwickelten, kryptographischen Software Pretty Good Privacy (PGP) auf. Aufgrund der großen Popularität von PGP und um anderen Programmen die Interoperabilität mit PGP zu ermöglichen, wurden die Grundfunktionen von PGP, sowie die verwendeten Datenstrukturen und Algorithmen erstmalig 1996 in Form des RFC 1991[33] veröffentlicht, damals basierend auf PGP 2.6.x. Diese Version von PGP bot nur eine eingeschränkte Auswahl an kryptographischen Algorithmen. Sie verwendete RSA für asymmetrische Verschlüsselung, IDEA[34] für symmetrische Verschlüsselung und MD5[35] als Hashfunktion.

Im Jahr 1998 wurde OpenPGP durch die IETF zum offenen Standard erklärt und im RFC 2440[36] formalisiert. Der RFC 2440 basiert auf der Nachfolgerversion PGP 5.x[37] und erweiterte den RFC 1991 in mehreren Bereichen. Zum einen wurden neue Algorithmen eingeführt. Als asymmetrische Algorithmen stehen in dieser OpenPGP Version, neben RSA, DSA und ElGamal zur Verfügung. Die Liste der symmetrischen Verfahren wurde unter anderem um 3DES, CAST[38] und Blowfish ergänzt. SHA-1 ersetzte MD5 als neue Standard-Hashfunktion. Als weitere Neuerung wurde das Protokoll um einige Datenstrukturen, wie z.B. neue Schlüssel- und Signaturversionen, erweitert. So ist es seit RFC 2440 beispielsweise möglich unterschiedliche asymmetrische Schlüssel für Verschlüsselung und Signieren zu verwenden. Die aktuelle Version der OpenPGP Spezifikation findet sich im RFC 4880 [CDF+07].

[33] Siehe http://www.ietf.org/rfc/rfc1991.txt

[34] IDEA steht für *International Data Encryption Algorithm*. Dabei handelt es sich um ein von Xuejia Lai und James L. Massey entwickeltes symmetrisches Verfahren. Die verwendete Schlüsselgröße beträgt 128 Bit.

[35] MD5 wird im RFC 1321 spezifiziert. Siehe auch http://tools.ietf.org/html/rfc1321 .

[36] Siehe auch http://www.ietf.org/rfc/rfc2440.txt

[37] Eine Kompatibilitätsliste der verschiedenen PGP Versionen findet sich unter http://www.spywarewarrior.com/uiuc/pgp-summ.htm .

[38] CAST, auch als CAST5 oder CAST128 bezeichnet, ist ein symmetrischer Algorithmus der von Carlisle Adams und Stafford Tavares entwickelt wurde. Eine Beschreibung des Verfahrens findet sich unter http://jya.com/cast.html .

Zu den Basisfunktionen von OpenPGP Anwendungen zählen:

- Verschlüsselung

  OpenPGP verwendet für die Datenverschlüsselung sowohl herkömmliche symmetrische Verfahren, als auch Hybridverfahren. Zusätzlich bietet es die Möglichkeit, eine Kombination von symmetrischen Verfahren zu nutzen. Dabei wird, wie bei den Hybridverfahren, ein symmetrischer Sitzungsschlüssel für die Verschlüsselung der eigentlichen Nachricht generiert. Der Sitzungsschlüssel selbst wird allerdings, im Gegensatz zum normalen Hybridverfahren, auch durch einen oder mehrere symmetrische Schlüssel, die aus Passphrasen generiert werden, geschützt. In der aktuellen OpenPGP Version ist es auch möglich, den Sitzungsschlüssel gleichzeitig mit symmetrischer und asymmetrischer Verschlüsselung zu verschlüsseln.

- Signieren

  Das Signieren von Daten funktioniert analog zu der Beschreibung in Kapitel 2. OpenPGP ermöglicht auch die Kombination aus der Verschlüsselung und dem Signieren von Daten. Im hybriden Verschlüsselungsschema wird dann entsprechend vor der symmetrischen Verschlüsselung die Signatur erzeugt und diese dann anschließend mit den Daten durch den Sitzungsschlüssel verschlüsselt.

- Kompression

  Um die Nachrichtengröße möglichst gering zu halten, komprimiert OpenPGP in der Regel Daten bevor sie verschlüsselt werden. Darauf wird bei der Beschreibung der OpenPGP-Pakete im Abschnitt 4.1.3 näher eingegangen.

- Radix-64 Konvertierung

  OpenPGP erlaubt die Kodierung von Daten in Form von 7-Bit ASCII-Zeichen[39]. Dies wird als Radix-64-Konvertierung bezeichnet und ausführlich im Abschnitt 4.3 erläutert.

Da im Rahmen der Entwicklung des OpenPGP Clients auch die OpenPGP Bibliothek in größerem Umfang neugestaltet wird (siehe Kapitel 6), stellt das weitere Kapitel die wichtigsten Elemente des OpenPGP Nachrichten Formats vor.

4.1 Aufbau von OpenPGP Nachrichten

OpenPGP verarbeitet und speichert Daten in Form von binären Nachrichten. Je nach verwendeter Funktion, kommen unterschiedliche Nachrichtentypen zum Einsatz. So werden verschlüsselte Daten zum Beispiel in einer *Encrypted Message* kodiert, komprimierte Daten hingegen in einer *Compressed Message* (eine Beschreibung der gültigen Nachrichtentypen findet sich in Abschnitt 4.1.4).

[39] ASCII steht für American Standard Code for Information Interchange. Dabei handelt es sich um eine 7-Bit-Zeichenkodierung, die aus 128 verschiedenen Zeichen besteht.

Eine Nachricht besteht aus einzelnen Komponenten, die als *Pakete* (Packet) bezeichnet werden. Ein Paket wiederum setzt sich aus einem *Kopf* (Packet Header) und einem *Körper* (Packet Body) zusammen. Auf unterster Ebene handelt es sich dabei um Folgen von Oktets[40]. Dies ist in Abbildung 4.1 dargestellt. Der Paketkopf besteht aus dem *Packet Tag*-Oktet und einer variablen Anzahl von Oktets, die die Länge des Paketkörpers kodieren. Der Packet Tag ist das erste Oktet des Paketkopfes und enthält Informationen über den Pakettyp und die Struktur des Kopfes. Diese Informationen sind in den einzelnen Bits des Packet Tags kodiert (siehe Abbildung 4.2).

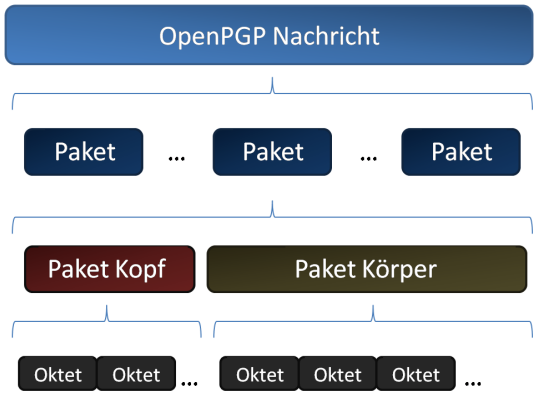

Abbildung 4.1: Struktur einer OpenPGP Nachricht

Die Interpretation der einzelnen Bits ist abhängig von dem verwendeten Paketformat. Der OpenPGP-Standard unterscheidet zwischen einem alten und einem neuen Paketformat, Bit 6 des Packet Tags indiziert dabei, welches der beiden Formate in einem Paket verwendet wird.

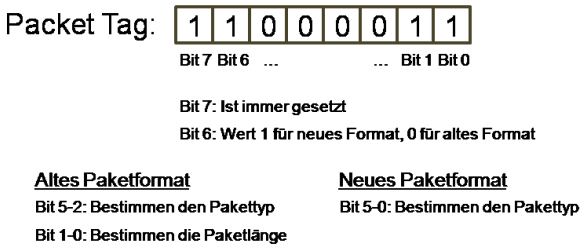
Packet Tag:
1 1 0 0 0 0 1 1
Bit 7 Bit 6 ... ... Bit 1 Bit 0
Bit 7: Ist immer gesetzt
Bit 6: Wert 1 für neues Format, 0 für altes Format
Altes Paketformat
Bit 5-2: Bestimmen den Pakettyp
Bit 1-0: Bestimmen die Paketlänge
Neues Paketformat
Bit 5-0: Bestimmen den Pakettyp

Abbildung 4.2: Struktur des Packet Tag

4.1.1 Pakete im alten Format (Old Format Packets)

Bei Paketen im alten Format enthalten die Bits 2-5 des Packet Tag den Pakettyp und die Bits 0-1 geben Auskunft über die Länge des Kopfes (und implizit auch über die

[40]Ein Oktet ist ein vorzeichenloser 8-Bit Wert. Der Wertebereich liegt zwischen 0 und 255. Dies entspricht dem Datentyp eines *unsigned byte*.

Länge des Körpers). Mit diesem Format können demnach nur 16 verschiedene Pakettypen unterschieden werden. Es wird von PGP in der Version 2.6.x genutzt und sollte nach [CDF+07] nur aus Gründen der Abwärtskompatibilität verwendet werden. Der Längentyp bestimmt wieviele Oktets nach dem Packet Tag, die Länge des Paketkörpers enthalten. Da für diese Information nur 2 Bits zur Verfügung stehen, gibt es demnach vier verschiedene Möglichkeiten (Werte 0 bis 3). Für die Werte 0 bis 2 steht die Länge des Paketkörpers in 1, 2 oder 4 Oktets. Der Header hat in diesen Fall also eine Gesamtlänge von 2, 3 bzw. 5 Oktets. Angenommen ein Paket ist vom Längentyp 0 und das eine Längenoktet nach dem Packet Tag enthält den Wert 150. Dann ist der Paketkörper 150 Oktets lang und das Paket hat eine Gesamtlänge von 152 Oktets. Pakete mit Längentyp 3 sind von unbestimmter Länge. In diesem Fall muss die Anwendung selbst das Ende des Pakets in Abhängigkeit vom aktuellen Kontext (z.B. dem Erreichen des Dateiendes) bestimmen. [CDF+07] raten von der Nutzung dieses Pakettyps ab und empfehlen Pakete definiter Länge oder Pakete im neuen Format einzusetzen. Die verschiedenen Varianten sind in Abbildung 4.3 dargestellt.

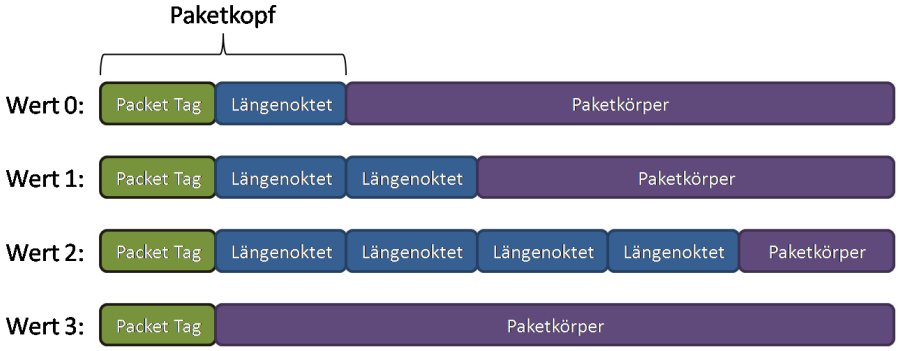

Abbildung 4.3: Verschiedene Längentypen im alten Paketformat

4.1.2 Pakete im neuen Format (New Format Packets)

Bei Paketen dieses Formats steht der Pakettyp in den Bits 0-5. Damit lassen sich maximal 64 Pakete unterscheiden. Ähnlich wie im alten Format lassen sich für Pakete diesen Typs vier verschiedene Längenkodierungen angeben. Diese werden nun allerdings anhand des Wertes des ersten Oktets nach dem Packet Tag unterschieden. Da die Kodierung bei neuen Paketen etwas komplexer ist, wird sie im Folgenden ausführlich dargestellt [CDF+07]:

- *1-Oktet Kodierung*: Bei der 1-Oktet-Kodierung entpricht die Länge des Paketkörpers dem Wert des ersten Oktets nach dem Packet Tag. Dieser Typ wird daran erkannt, dass das erste Oktet einen Wert kleiner als 192 hat. Ein zu kodierender Paketkörper kann in dieser Variante also maximal 191 Oktets lang sein. Der Header hat eine Länge von insgesamt 2 Oktets.

- *2-Oktet Kodierung*: Bei diesem Typ ist die Längeninformation in 2 Oktets gespeichert, der Paketkopf besteht aus insgesamt 3 Oktets. Die Länge des Paketkörpers wird anhand der Formel

$$((1tes\_Oktet - 192) << 8) + 2tes\_Oktet + 192$$

  berechnet. Die 2-Oktet-Kodierung wird daran erkannt, dass der erste Oktet-Wert im Bereich von 192 bis einschließlich 223 liegt. Mit der angegebenen Formel lassen sich Paketkörper mit einer Länge von 192 bis hin zu 8383 Oktets kodieren.

- *5-Oktet Kodierung*: Ein Paket verwendet die 5-Oktet-Kodierung wenn das erste Oktet den Wert 255 hat. Die nachfolgenden vier Oktets enthalten die Länge des Paketkörpers. Sie wird anhand der Formel

$$(2tes\_Oktet << 24) \mid (3tes\_Oktet << 16) \mid (4tes\_Oktet << 8) \mid 5tes\_Oktet$$

  berechnet. Der Paketkopf hat eine Gesamtlänge von 6 Oktets. Es lassen sich Paketkörper mit einer Länge bis zu 4.294.967.295 Oktets kodieren.

- *Partielle Kodierung* (partial body length): Partielle Kodierung wird verwendet, wenn die Gesamtpaketgröße zu Beginn der Nachrichtenerstellung unbekannt ist. Dies kann zum Beispiel bei der Kompression von Daten vorkommen. Das Paket wird in diesem Fall in einzelnen Teilen, sogenannten *chunks*, kodiert. Je ein partieller Paketkopf kodiert die Länge eines Paketteils, gefolgt von dem Paketteil selbst. Dabei ist zu beachten das nur der erste partielle Kopf einen Packet Tag enthält, alle anderen Paketteile werden lediglich mit der Längenkodierung versehen. Dieses Verfahren wird solange wiederholt, bis das Ende des Pakets erreicht wird. Der abschließende finale Paketteil muss mit einer 1-, 2- oder 5-Oktet-Kodierung versehen werden. Ein einfaches Beispiel für eine partielle Kodierung ist in Abbildung 4.4 dargestellt. Die Länge des Paketteils wird über die Formel

$$\text{Länge} = 1 << (1tes\_Oktet \ \& \ 31)$$

  berechnet. Es können somit Längen zwischen $2^0 = 1$ Oktet und $2^{30} = 1.073.741.824$ Oktets kodiert werden. Bei partiell kodierten Paketen hat das erste Oktet einen Wert von 224 bis einschließlich 254. Diese Form der Kodierung darf allerdings ausschließlich für Datenpakete verwendet werden. Diese werden im folgenden Abschnitt 4.1.3 näher beschrieben.

Abbildung 4.4: Beispiel für eine partielle Kodierung eines Datenpakets

4.1.3 Pakettypen

Im vorherigen Kapitel wurde erläutert, dass der Typ eines Pakets über den Packet Tag bestimmt wird. RFC 4880 definiert zur Zeit die in Tabelle 4.1 aufgeführten Pakete. Einige der Pakete, wie z.B. Public Key Packet oder Signature Packet, haben eine interne Versionsnummer, so dass es durchaus verschiedene Ausprägungen dieser Pakete geben kann. Die angegebene Paketliste kann zudem, mittels der dafür zur Verfügung stehenden Tags 60-63, um selbst entworfene Pakettypen erweitert werden. Diese können dann innerhalb einer eigenen OpenPGP-Implementation verwendet werden.

Packet Tag	**Pakettyp**
0	Reserviert
1	Public Key Encrypted Session Key Packet
2	Signature Packet
3	Symmetric-Key Encrypted Session Key Packet
4	One-Pass Signature Packet
5	Secret Key Packet
6	Public Key Packet
7	Secret Subkey Subpacket
8	Compressed Data Packet
9	Symmetrically Encrypted Data Packet
10	Marker Packet
11	Literal Data Packet
12	Trust Packet
13	User ID Packet
14	Public Subkey Packet
17	User Attribute Packet
18	Sym. Encrypted and Integrity Protected Data Packet
19	Modification Detection Code Packet
60 - 63	Private oder experimentelle Pakete

Tabelle 4.1: OpenPGP Pakettypen. Quelle: [CDF+07].

Die wichtigsten Pakettypen werden im Folgenden kurz vorgestellt. Für eine ausführliche Beschreibung wird auf [CDF+07] verwiesen.

4.1.3.1 Datenpakete (Data Packets)

Pakete diesen Typs speichern die transportierten Daten einer Nachricht. Es gibt drei verschiedene Datenpakettypen, komprimierte (Compressed Data Packet), symmetrisch verschlüsselte (Symmetrically Encrypted Data Packet) und literale Datenpakete (Literal Data Packet). Der Typ des Pakets gibt dabei Aufschluss über die auf die Daten angewendete Funktion und wie die enthaltenen Daten interpretiert werden müssen. Komprimierte und verschlüsselte Datenpakete enthalten in der Regel andere Pakete oder komplette Nachrichten, in jedem Fall aber ein literales Datenpaket, welches die Rohdaten der Nachricht beinhaltet. Wie im vorherigen Abschnitt erwähnt, dürfen ausschließlich Datenpakete eine partielle Kodierung im Paketkopf verwenden.

Compressed Data Packet
Ein *Compressed Data Packet* speichert Daten in komprimierter Form. OpenPGP unterstützt zur Zeit die Kompressionsverfahren ZIP[41], ZLIB[42] und BZip2[43]. Ein solches Paket besteht aus einem Oktet, welches den benutzten Kompressionsalgorithmus identifiziert, sowie den komprimierten Daten selbst.

Symmetrically Encrypted Data Packet/Symmetrically Encrypted Integrity Protected Data Packet
Diese Pakete speichern Daten, die mit einem symmetrischen Sitzungsschlüssel verschlüsselt wurden. Beim *Symmetrically Encrypted Data Packet* handelt es sich um die ältere Basisversion dieses Paketstyps. Es speichert ausschließlich die verschlüsselten Daten. Das *Symmetrically Encrypted Integrity Protected Data Packet* erweitert dieses Basispaket um einen zusätzlichen Integritätsschutz in Form eines *Modification Detection Code Packets*. Das Modification Detection Code Packet enthält einen mit SHA-1 berechneten Hashwert über den zu verschlüsselnden Klartext, sowie über den fixen Paketkopf des Modification Detection Code Packets selbst. Das Paket wird zusammen mit dem Klartext verschlüsselt und ermöglicht somit einem Empfänger die Prüfung der Integrität des Paketinhalts.

Literal Data Packet
Ein literales Datenpaket enthält die Rohdaten einer Nachricht, wie z.B. eine Grafik oder einen Text. Zusätzlich beinhaltet es Metainformationen über die gespeicherten Daten, beispielsweise ob es sich um binäre oder textbasierte Daten handelt, ggf. einen Dateinamen oder das Erstellungs- bzw. Modifikationsdatum der gespeicherten Daten. Der Inhalt eines literalen Datenpakets kann zudem als sensitiv markiert werden, wenn der Name `_CONSOLE` als Dateiname verwendet wird. In diesem Fall sollte "das empfangende Programm die Daten mit größerer Vorsicht verarbeiten und möglicherweise das Speichern der Daten auf Festplatte vermeiden" [CDF+07]. Enthält das literale Datenpaket Text, so wird dieser vor der Speicherung in das kanonische OpenPGP-Format überführt, um

[41] Siehe auch http://www.ietf.org/rfc/rfc1951.txt
[42] Siehe auch http://www.ietf.org/rfc/rfc1950.txt
[43] BZip2 ist ein von Julian Seward entwickelter Kompressionsalgorithmus. Siehe auch http://www.bzip.org/

Probleme mit betriebssystemabhängigen Zeilenumbrüchen zu umgehen. Dabei werden alle Zeilenumbrüche einheitlich durch die Zeichen `<CR><LF>`[44] ersetzt.

4.1.3.2 Sitzungsschlüsselpakete (Session Key Packets)

Sitzungsschlüsselpakete werden verwendet, um den Sitzungsschlüssel selbst mit symmetrischer oder asymmetrischer Kryptographie zu schützen Sie werden unterschieden in *Public Key Encrypted Session Key Packets* und *Symmetric-Key Encrypted Session Key Packets* und werden einem symmetrisch verschlüsseltem Datenpaket vorangestellt. Für ein Symmetrically Encrypted Data Packet ist die Verwendung von Sitzungsschlüsselpaketen optional. Wird dem Datenpaket also kein Sitzungsschlüsselpaket vorangestellt, geht OpenPGP implizit davon aus, dass IDEA als symmetrischer Algorithmus verwendet wird. Der Sitzungsschlüssel berechnet sich in diesem Fall aus dem MD5-Hashwert über einer Passphrase. Aus Sicherheitsgründen sollte jedoch auf diesen Mechanismus verzichtet werden. Der neueren Variante des Datenpakets, dem Symmetrically Encrypted Integrity Protected Data Packet, muss daher nach [CDF+07] auch mindestens ein Sitzungsschlüsselpaket vorangehen.

Public Key Encrypted Session Key Packets
Ein *Public Key Encrypted Session Key Packet* enthält den, mit dem öffentlichen Schlüssel eines Nachrichtenempfängers verschlüsselten, Sitzungsschlüssel. Pro Nachrichtenempfänger wird je ein solches Paket generiert und vor das symmetrisch verschlüsselte Datenpaket gestellt. Um einen öffentlichen Schlüssel zu identifizieren wird im Public Key Encrypted Session Key Packet die Kennung (ID) des jeweiligen Schlüssels (siehe hierzu Abschnitt 4.2), sowie der verwendete asymmetrische Algorithmus gespeichert. Es können sowohl RSA, als auch ElGamal benutzt werden.

Symmetric-Key Encrypted Session Key Packets
Ein *Symmetric-Key Encrypted Session Key Packet* schützt den Sitzungsschlüssel mit einem, aus einer Passphrase generierten, symmetrischen Schlüssel. Pro Passphrase, mit der der Sitzungsschlüssel verschlüsselt werden soll, wird dem symmetrisch verschlüsselten Datenpaket ein Symmetric-Key Encrypted Session Key Packet vorangestellt.

Um einen symmetrischen Schlüssel aus einer Passphrase zu erzeugen, verwendet OpenPGP sogenannte *String-To-Key-Specifier*[45] (S2K), die den Schlüssel über eine Hashfunktion berechnen. OpenPGP unterscheidet verschiedene S2K Typen. Im einfachsten Fall wird lediglich die Hashfunktion auf die Passphrase angewendet. Aus Sicherheitsgründen gibt es jedoch auch (bevorzugte) Varianten, die der Passphrase einen zufälligen Wert, das sogenannte *Salz* (salt) hinzufügen, bzw. eine iterierte Hashberechnung durchführen, um mögliche Angriffe zu verhindern. Ein S2K kann nun in einem Symmetric-Key Encrypted Session Key Packet auf zweierlei Art Anwendung finden. Enthält das Sitzungsschlüssel-

[44] `CR` oder `\r` steht für den Wagenrücklauf (Carriage Return), der im ASCII-Zeichensatz durch das Zeichen 13 dargestellt wird. `LF` oder `\n` steht für Linefeed und ist Zeichen 10 des ASCII-Zeichensatzes.
[45] Siehe hierzu [CDF+07] Sektion 3.7.

paket den Sitzungsschlüssel selbst, so wird der S2K verwendet, um den Sitzungsschlüssel zu entschlüsseln. Alternativ ist es auch möglich nur den S2K im Sitzungsschlüsselpaket zu speichern. In diesem Fall generiert der S2K, den für das Datenpaket verwendeten Sitzungsschlüssel.

Das Sitzungsschlüsselpaket enthält neben dem S2K ein Oktet, das angibt, mit welchem symmetrischen Algorithmus der Sitzungsschlüssel verschlüsselt wurde. Wenn zusätzlich der Sitzungsschlüssel selbst vorhanden ist, so wird diesem ebenfalls ein Oktet vorangestellt, welches angibt, mit welchem Algorithmus das nachfolgende Datenpaket verschlüsselt wurde. Dieses Oktet wird, wie auch der Sitzungsschlüssel, mit dem S2K verschlüsselt. Sitzungsschlüssel und Daten können somit unterschiedliche Algorithmen nutzen. OpenPGP unterstützt zur Zeit die Verfahren IDEA, TripleDES, CAST, Blowfish, AES und Twofish. Die vollständige Struktur eines Symmetric-Key Encrypted Session Key Packets ist in Abbildung 4.5 dargestellt.

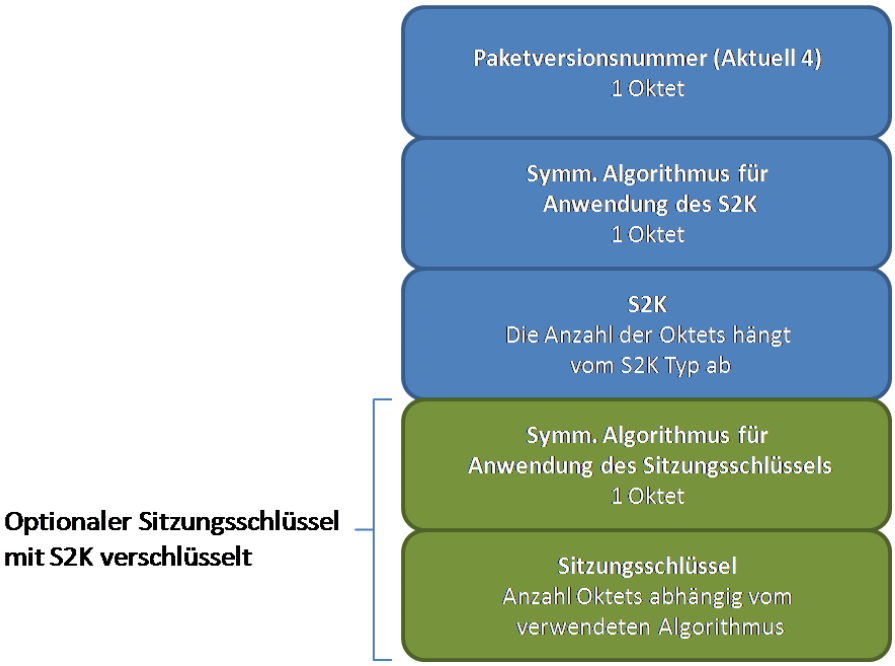

Abbildung 4.5: Struktur eines Symmetric-Key Encrypted Session Key Packet

4.1.3.3 Schlüsselmaterialpakete (Key Material Packets)

Bevor auf die verschiedenen zugehörigen Schlüsselmaterialpakete eingegangen wird, muss zunächst der Begriff des *OpenPGP Schlüssels* (OpenPGP Key) eingeführt werden. Diese werden ausführlich in Abschnitt 4.2 diskutiert. Ein OpenPGP Schlüssel erfüllt in OpenPGP die Funktion eines Zertifikats. Er enthält unter anderem ein primäres asymmetrisches Schlüsselpaar (primary key pair) und (je nach Version) eine Menge von asymmetrischen Unterschlüsselpaaren (subkey pairs oder subkeys), die dem Primärschlüssel

zugeordnet sind. Dabei wird das primäre Schlüsselpaar für das Signieren und die untergeordneten Schlüsselpaare für die Verschlüsselung von Daten genutzt. Die Schlüsselmaterialpakete stellen einen integralen Bestandteil eines OpenPGP Keys dar. Sie enthalten die öffentlichen oder privaten Schlüssel eines Schlüsselpaars. OpenPGP unterscheidet zwischen *Public Key Packet*, *Public Subkey Packet*, *Secret Key Packet* und *Secret Subkey Packet*.

Public Key Packet/Public Subkey Packet
Ein Public Key Packet speichert den öffentlichen Schlüssel eines primären Schlüsselpaars, ein Public Subkey Packet den öffentlichen Schlüssel eines untergeordneten Schlüsselpaars. Die beiden Pakettypen sind von der Struktur her identisch, sie unterscheiden sich nur durch die Bedeutung des öffentlichen Schlüssels. Public Key und Subkey Packets existieren in zwei verschiedenen Versionen. Version 3 wurde mit PGP 2.6 eingeführt, die aktuellen Pakete der Version 4 in PGP 5.0. Das Format eines Paketes der Version 4 ist in Abbildung 4.6 dargestellt. Die Public-Key Parameter (das eigentliche "Schlüssel-

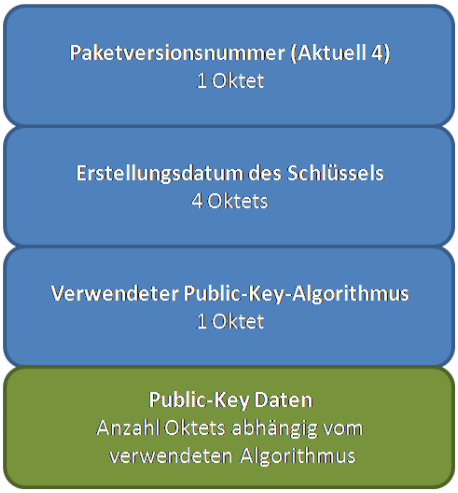

Abbildung 4.6: Struktur eines Public Key/Subkey Packet der Version 4

material") werden in Form von sogenannten *Multiprecision Integers* (MPI) gespeichert. Bei MPIs handelt es sich um große positive Ganzzahlen (unsigned integer), denen eine Längenangabe bestehend aus 2 Oktets vorangestellt wird Die Längenangabe kodiert die Länge des eigentlichen Zahlenwertes in Bits. Public Key/Subkey Packets der Version 3 unterstützen aussschließlich RSA[46], Pakete der Version 4 RSA, ElGamal und DSA. Da die Algorithmen unterschiedliche Parameter verwenden, ist die Anzahl der MPIs im Public Key/Subkey Packet abhängig vom verwendeten Verfahren.

Ein öffentlicher Schlüssel wird in OpenPGP über eine *Schlüsselidentität* (Key ID) oder

[46]Dies ist bedingt durch die Beschränkungen hinsichtlich der zur Verfügung stehen Algorithmen in PGP 2.6.x. Siehe auch die Einleitung dieses Kapitels.

einen *Fingerabdruck* (Fingerprint) identifiziert[47]. Die Berechnung von Schlüsselidentität und Fingerabdruck unterscheidet sich, in Abhängigkeit von der Version des Public Key/-Subkey Packet. Die Identität eines Schlüssels in der Version 3 sind die unteren 64 Bit des Public Key Modulus[48] des RSA Schlüssels. Die Key ID hat somit eine Länge von 8 Oktets. Der Fingerabdruck ist der MD5-Hashwert über alle MPIs des öffentlichen Schlüssels. Dabei ist zu beachten, dass die Längeninformationen der einzelnen MPIs nicht in die Berechnung des Hashwerts mit eingehen. Bei einem Schlüssel der Version 4 ist der Fingerabdruck der SHA1-Hashwert über das gesamte Schlüsselpaket, die Schlüsselidentität sind die unteren 64 Bit des Fingerabdrucks. Schlüsselpakete der Version 3 gelten als veraltet. [CDF+07] weisen insbesondere auf folgende Schwächen bei Paketen dieser Version hin:

- Kollision bei Schlüsselidentitäten

  "Es ist relativ einfach einen [...] Schlüssel zu generieren, der die gleiche Schlüsselidentität wie ein anderer Schlüssel hat, da die Schlüsselidentität einfach nur die unteren 64 Bits des Public-Key Modulus sind." [CDF+07]

- Kollision bei Fingerabdrücken

  "Da der Fingerabdruck das Schlüsselmaterial, aber nicht die Länge [...] [mit einbezieht], gibt es ein erhöhtes [...] [Risiko] für [...] [Kollisionen]." [CDF+07]

- Sicherheit von MD5

  MD5 gilt nicht mehr als kollisionsresistent, daher sollte diese Funktion nur mit Vorsicht verwendet werden [Buc03][49].

Secret Key Packet/Secret Subkey Packet
Ein Secret Key/Subkey Packet besteht aus einem Public Key/Subkey Packet gefolgt von den Daten des zugehörigen privaten Schlüssels, die analog zum Public Key/Subkey Packet in Form von MPIs gespeichert werden. Das Paket enthält also sowohl den öffentlichen Schlüssel, als auch den privaten Schlüssel eines Schlüsselpaars. Die Version des Pakets (3 oder 4), wie auch Schlüsselidentität und Fingerabdruck, wird dabei durch die Version des enthaltenen öffentlichen Schlüsselpakets bestimmt. Der private Schlüssel wird, ähnlich wie beim Symmetrically Encrypted Session Key Packet, im Allgemeinen durch einen symmetrischen Algorithmus geschützt. Der symmetrische Schlüssel wird über einen S2K generiert. Enthält das Paket keinen S2K verwendet OpenPGP automatisch MD5. Aufgrund der oben erwähnten Sicherheitsproblematik sollte daher immer ein S2K verwendet werden. OpenPGP berechnet zudem eine Prüfsumme oder einen SHA1-Hashwert über die privaten Schlüsseldaten, um eine einfache Integritätsprüfung zu ermöglichen. Bei Paketen der Version 3 wird dieser Prüfwert im Klartext gespeichert, in Version 4 hingegen mit dem privaten Schlüsselmaterial verschlüsselt.

[47] Wird der öffentliche Schlüssel in anderen Paketen referenziert, geschieht dies meist anhand der Key ID.

[48] Für eine konkrete Beschreibung der Parameter des RSA Verfahrens siehe z.B. [Buc03].

[49] Vergleiche hierzu die Originalreferenz aus [Buc03] H. Dobbertin. The status of MD5 after a recent attack. CryptoBytes, 2(2):1-6, 1996.

4.1.3.4 Benutzerpakete (User Packets)

In Benutzerpaketen werden Informationen über die Identität eines Schlüsselbesitzers gespeichert. Hierfür definiert der OpenPGP-Standard zwei Pakettypen, das *User ID Packet* und das *User Attribute Packet*. Da ein Benutzer durchaus mehrere solche Identitäten bzw. Attribute haben kann, lässt sich eine ID oder ein Attribut als *primär* markieren. Diese Information ist, wenn vorhanden, im zugehörigen Signaturpaket gespeichert, dass das Paket an den Schlüssel bindet.

User ID Packet
Ein User ID Packet stellt eine textuelle Beschreibung des Inhabers eines öffentlichen Schlüssels dar. Es besteht, wie in Kapitel 3.2 dargestellt, aus einem Namen, sowie einer E-Mail-Adresse und wird im Allgemeinen in der Form `Name <vor.nachname@domain.tld>` angegeben.

User Attribute Packet
Während ein User ID Packet auf eine textuelle Beschreibung des Schlüsselbesitzers beschränkt ist, wurde das User Attribute Packet für alternative Beschreibungsformen vorgesehen. Es kann aus ein oder mehreren untergeordneten Paketen (sub packets) bestehen, die verschiedene Attribute des Schlüsselbesitzers enthalten können. Jeder Unterpakettyp entspricht dabei einem möglichen Attribut. Der einzige, momentan definierte Unterpakettyp, ist ein Bild im *JPEG File Interchange Format*[50] (JFIF).

4.1.3.5 Signaturpakete (Signature Packets)

Pakete diesen Typs enthalten von OpenPGP berechnete Signaturen, sowie allgemeine Informationen (wie z.B. verwendetes Hashverfahren, Schlüsselidentität, asymmetrischer Algorithmus etc.), die für die Verifikation bzw. Verarbeitung der Signatur von Bedeutung sind. OpenPGP unterscheidet verschiedene Signaturtypen, je nachdem für welche Art von Daten die Signatur erzeugt wurde. So ist zum Beispiel eine Signatur über ein binäres Dokument, Textdokument oder auch über eine Benutzeridentität möglich[51]. Der Signaturtyp wird über ein Oktet im Paket identifiziert. Dies ist insbesondere für die Verifikation relevant, da für die verschiedenen Typen ggf. unterschiedliche Berechnungen durchgeführt werden müssen[52]. Analog zu den Schlüsselpaketen (siehe oben), gibt es Signaturpakete in den Versionen 3 und 4. Abbildung 4.7 zeigt den Aufbau eines Signaturpakets der Version 4. Pakete diesen Typs können, neben den Basisinformationen, *Signaturunterpakete* (Signature Subpacket) enthalten, die die Angabe von zusätzlichen Informationen ermöglichen [CDF+07]. Dazu zählen unter anderem die Angabe von Signaturerstellungsdatum, Signaturablaufdatum, ID des Widerufsschlüssels (für den Fall, dass die Signatur revoziert wurde) etc.[53]. Ein Signaturunterpaket hat, wie die normalen Pakettypen auch, einen Kopf (header) und einen Körper (body). Der Header kodiert die

[50] Siehe http://www.w3.org/Graphics/JPEG/jfif3.pdf
[51] Eine Liste aller Signaturtypen findet sich in [CDF+07] Sektion 5.2.1.
[52] Für eine Bescheibung der einzelnen Berechnungsverfahren siehe auch [CDF+07] Sektion 5.2.4.
[53] Für eine Liste aller definierte Signature Subpackets siehe [CDF+07] Sektion 5.2.3.1.

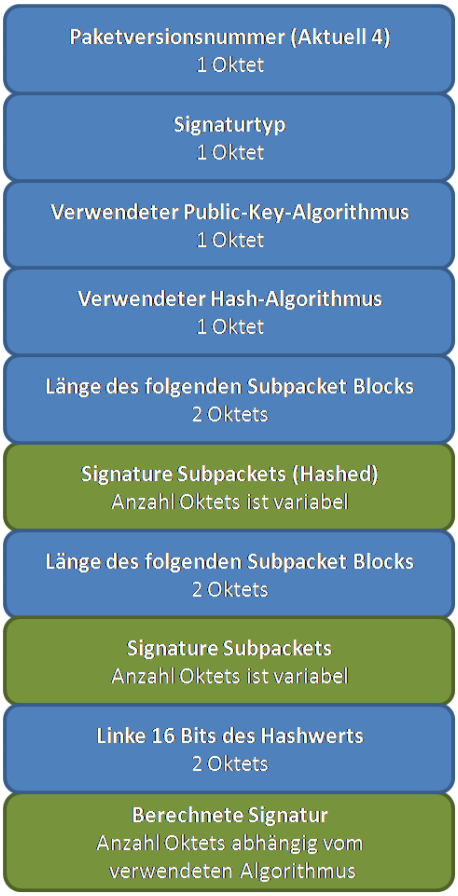

Abbildung 4.7: Struktur eines Signature Packet der Version 4

Länge des Pakets und enthält den Pakettyp. Die Längenkodierung ist dabei identisch zu dem in Abschnitt 4.7 beschriebenem neuen Paketformat. Allerdings können Signaturunterpakete keine partielle Kodierung verwenden. Bei der Erstellung einer Signatur geht immer ein Teil des Signaturpakets in die Berechnung des zu signierenden Hashwerts mit ein. Daher gibt es zwei Bereiche mit Unterpaketen, von denen nur einer mit "gehashed" wird. In diesem Bereich sollten daher sensitive Information abgelegt werden. Nach den Subpackets folgen zwei Oktets, die die linken 16 Bits des signierten Hashwerts speichern. Dies ermöglicht einen einfachen Test der Gültigkeit der Signatur. Abschließend enthält das Paket die Signatur selbst. Diese wird, wie auch das Schlüsselmaterial im Public Key/Subkey Packet, in Form von MPIs gespeichert. Die Anzahl der MPIs ist wiederum abhängig vom Signaturverfahren. Sowohl Version 3, als auch Version 4 unterstützen RSA und DSA.

4.1.4 Nachrichtentypen

Die Semantik einer OpenPGP Nachricht, wird durch die Pakete bestimmt, aus denen sie zusammengesetzt ist. Abbildung 4.8 zeigt die momentan in [CDF+07] definierten Nach-

richtentypen und beschreibt die Regeln, die bei der Nachrichtenerstellung eingehalten werden müssen. Alle anderen Paketkombinationen stellen keine gültige Nachricht dar. Wie in der Abbildung ersichtlich, sind die Regeln zum Teil rekursiv, dass heisst eine

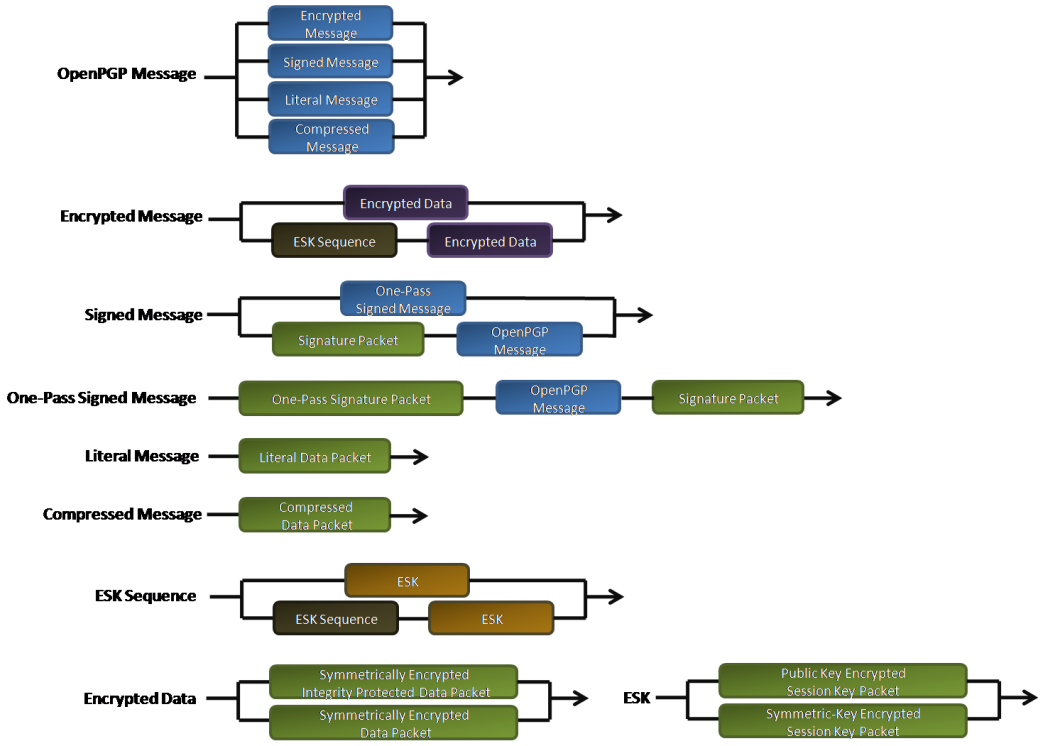

Abbildung 4.8: Syntaktische Regeln für die Komposition von OpenPGP Nachrichten

Nachricht kann durchaus eine andere Nachricht enthalten. Dadurch ist eine Verkettung der OpenPGP Basisfunktionen möglich, z.B. um Daten erst zu signieren, dann zu komprimieren und anschließend zu verschlüsseln. Bei einer solchen Verschachtelung ist die innerste Nachricht immer eine Literal Message, die die eigentlichen Nutzdaten, wie zu signierenden Text, oder eine zu verschlüsselnde Graphik usw. enthält (siehe Abbildung 4.9). Die rekursive Natur der Nachrichten spielt insbesondere bei der Umsetzung der Nachrichtenverarbeitung im OpenPGP Client eine große Rolle (siehe Kapitel 6).

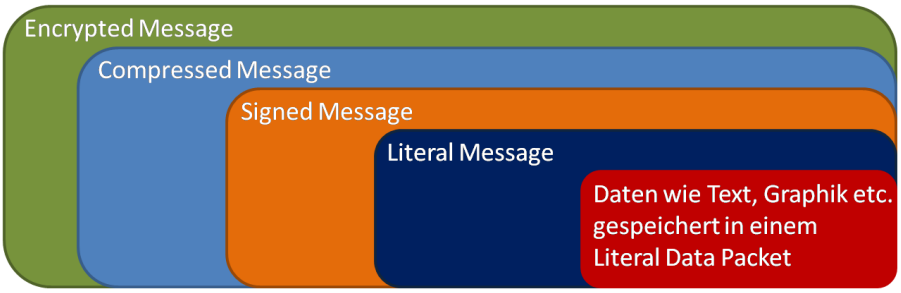

Abbildung 4.9: Beispiel für eine mehrfach verschachtelte OpenPGP Nachricht

4.2 OpenPGP Schlüssel

Ein *OpenPGP Schlüssel* (OpenPGP Key) oder auch *transferierbarer Schlüssel* (transferable key) erfüllt in OpenPGP die Funktion eines Zertifikats, vergleichbar mit dem in Kapitel 3 vorgestellten X.509-Zertifikaten. Er enthält ein oder mehrere zu einem Besitzer gehörende Schlüsselpaare (siehe Kapitel 3.2), Benutzerinformationen, die durch Signaturen an die Schlüssel gebunden werden, sowie die zu den Schlüsselpaaren gehörenden Signaturen selbst (dazu zählen z.B. auch Widerufssignaturen und Signaturen, die Unterschlüssel an Hauptschlüssel binden). Je nachdem, ob nur die öffentlichen Schlüssel eines Besitzers oder auch die privaten Schlüssel Teil des OpenPGP Schlüssels sind, wird er entweder als *Transferable Public Key* oder als *Transferable Secret Key* beschrieben. OpenPGP Schlüssel werden normalerweise in Form eines Schlüsselbunds (key ring) auf der Festplatte gespeichert. Die privaten Schlüssel werden im *geheimen Schlüsselbund* (secret key ring) abgelegt, die öffentlichen Schlüssel im *öffentlichen Schlüsselbund* (public key ring). Die konkrete Umsetzung des Schlüsselbunds wird in der OpenPGP Spezifikation allerdings offengelassen. PGP und GnuPG verwenden beispielsweise lokale Dateien, der OpenPGP Client hingegen eine lokale Datenbank (siehe Kapitel 6).

OpenPGP Schlüssel können in Version 3 oder 4 vorkommen. Dies hängt von der Paketversion der enthaltenen Schlüsselpaare ab. Nur Schlüssel der Version 4 können Unterschlüsselpaare haben. In diesem Fall muss der Primärschlüssel ein Signaturschlüssel sein, während die Unterschlüssel eine beliebige Funktion erfüllen können. Generell werden diese aber für die Verschlüsselung von Daten verwendet. Die Struktur eines OpenPGP Public Keys in der Version 4 ist in Abbildung 4.10 dargestellt. Ein OpenPGP Secret Key enthält statt den Public Key/Subkey Paketen entsprechend Secret Key/Subkey Pakete. Wie in der Abbildung ersichtlich, muss mindestens eine Benutzeridentität pro Schlüssel definiert sein. Hinter jeder Identität bzw. hinter jedem Attribut folgt eine Liste von Signaturen, die als Zertifikate[54] (certificates) bezeichnet werden. Man unterscheidet dabei zwischen, vom Schlüsseleigner *selbstsignierten Zertifikaten* (self-signed certificates) und gewöhnlichen Zertifikaten, die von anderen Schlüsselbesitzern ausgestellt werden können, um ihr Vertrauen in die Bindung zwischen Schlüsselinhaber und Schlüssel auszudrücken (siehe hierzu auch die Beschreibung des Netz des Vertrauens in Kapitel 3.2). Nach den Benutzerinformationen folgen die optionalen Unterschlüssel. Jeder Unterschlüssel muss durch eine Signatur an den Primärschlüssel gebunden werden. Optional folgt nach dieser Bindungssignatur eine Widerufsignatur, falls der Unterschlüssel revoziert wurde.

4.3 Radix-64 Konvertierung

In Abschnitt 4.1 wurde erläutert, dass OpenPGP-Nachrichten in einem Binärformat, bestehend aus 8-Bit-Oktets verarbeitet und gespeichert werden. Die *Radix-64*-Konvertierung wird verwendet, um diese 8-Bit-Daten als 7-Bit *ASCII*-Zeichen zu kodieren. Dies ist beispielsweise notwendig, um verschlüsselte OpenPGP-Nachrichten, die per

[54] Der Begriff des Zertifikats ist in der OpenPGP Terminologie leider mehrdeutig.

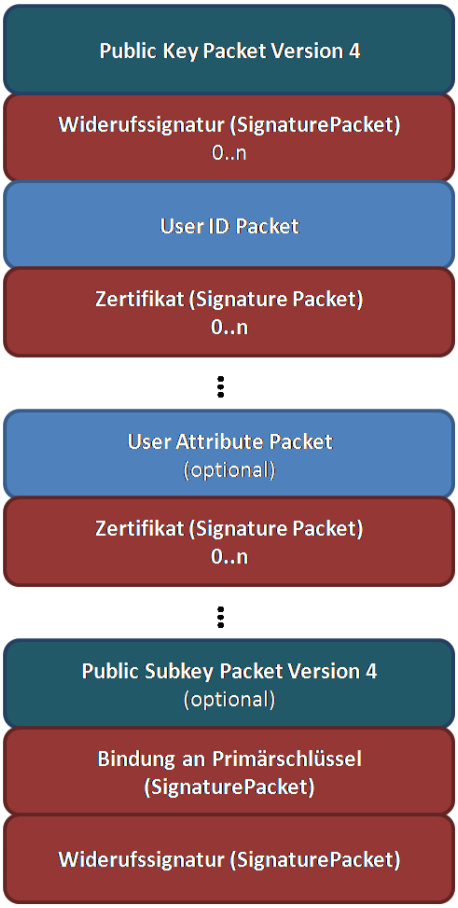

Abbildung 4.10: Struktur eines OpenPGP Schlüssels Version 4

E-Mail versendet werden sollen, vor Datenverlust beim Transport zu schützen, da es E-Mail-Systeme gibt, die ausschließlich ASCII-Zeichen verarbeiten können. Das Verfahren wird daher auch als *ASCII Armor* bezeichnet. Die Radix-64-Konvertierung ist identisch zur *Base64*-Konvertierung[55], erweitert diese aber um eine 24-Bit-Prüfsumme, die an die kodierten Daten angehängt wird. Die Prüfsumme wird verwendet, um eventuell auftretende Übertragungsfehler festzustellen.

Wie in Abbildung 4.11 dargestellt, kodiert Base64 je drei Oktets des Eingabestroms zu vier 6-Bit-Werten zwischen 0 und 63. Diese Werte werden anhand der in Abbildung 4.2 gezeigten Tabelle in ASCII-Zeichen konvertiert. Die Ausgabe der kodierten Zeichen erfolgt dabei zeilenweise, die maximale Zeilenlänge ist auf 76 Zeichen beschränkt. Enthält der Eingabestrom eine nicht durch drei teilbare Anzahl an Oktets, wird die Eingabe pro fehlendem Oktet, mit einem Oktet der Wertigkeit 0, aufgefüllt. Dies kann je nach Eingabelänge keinmal, einmal oder zweimal erfolgen. Die kodierte Ausgabe hingegen wird pro hinzugefügtem 0-Oktet um das Füllzeichen = ergänzt. Die Füllzeichen signalisieren

[55]Die in Radix-64 genutzte Base64-Konvertierung basiert auf dem RFC 1421 und wird im MIME-Standard (Multipurpose Internet Mail Extensions) verwendet, der in RFC 2045 definiert ist.

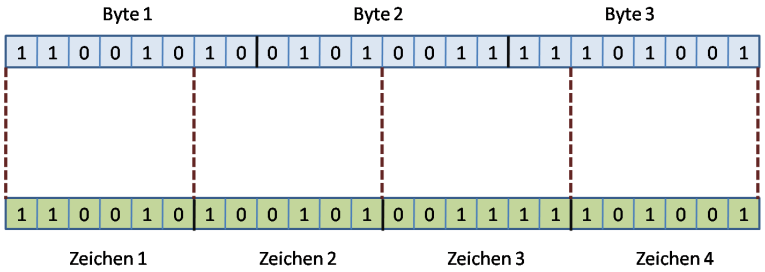

Abbildung 4.11: Prinzip der Base64-Kodierung

einer Dekodiersoftware die Anzahl von Fülloktets. Die Prüfsumme wird während des

Wert	Zeichen	Wert	Zeichen	Wert	Zeichen	Wert	Zeichen
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(Füll)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Tabelle 4.2: Base64 Umsetzungstabelle. Quelle: [CDF+07].

Konvertierungsprozesses über die Eingangsdaten berechnet. Es handelt sich dabei um eine *zyklische Redundanzprüfung*[56] (cyclic redundancy check, CRC). Da die Prüfsumme selbst Base64-kodiert wird, verwendet OpenPGP eine 24-Bit-Prüfsumme (CRC-24), da diese dann ohne Füllbytes auf vier ASCII-Zeichen abgebildet werden kann. Sie wird, zusammen mit einem vorangehenden =, in die Folgezeile nach den kodierten Eingangsdaten

[56] CRC-Verfahren basieren auf Polynomdivision. Der Eingabestrom wird dabei als Polynom aufgefasst, welcher modulo durch ein vordefiniertes *Generatorpolynom* (generator polynom) geteilt wird. Der übriggebliebe Rest bildet dann die Prüfsumme, die beim Empfänger der Daten auf Korrektheit validiert werden kann. CRC-24 bildet beispielsweise auf eine 24-Bit Prüfsumme ab. Für Details siehe auch [MF06].

geschrieben.

Die Konvertierung des Textes *Ein Beispiel für Radix-64-Konvertierung* liefert beispielsweise folgendes Ergebnis:

```
RWluIEJlaXNwaWVsIGbDvHIgUmFkaXgtNjQtS29udmVydGllcnVuZw==
=Jii9
```

OpenPGP stellt den Radix-64-kodierten Daten Metainformationen, sogenannte *Armor Header*, voran. Diese werden in einzelnen Zeilen angegeben. Die erste Zeile gibt Aufschluss über den Typ der kodierten Nachricht und markiert den Beginn des Armor Header Blocks. Sie wird als *Armor Header Line* bezeichnet und von jeder Seite von fünf Bindestrichen umgeben. In den Folgezeilen stehen die Armor Header, welche ggf. Informationen über den Ersteller, oder Parameter, welche für die Verarbeitung der Nachricht relevant sind, enthalten. Sie werden in Form von Schlüssel/Wert-Paaren angegeben, die durch einen Doppelpunkt gefolgt von einem Leerzeichen getrennt werden. Danach werden die Radix-64-kodierten Daten geschrieben. Abschließend folgt eine Zeile als Anhang, der *Armor Tail Line*, die wiederum durch den Nachrichtentyp bestimmt wird. Die Metainformationen schließen also die kodierte Nachricht ein.

Der komplette Aufbau einer Radix-64-konvertierten Nachricht ist also wie folgt [CDF+07]:

- Armor Header Line, die den Nachrichtentyp beschreibt
- Mehrere Armor Header Zeilen mit Informationen über die Nachricht
- Eine Leerzeile
- Base64-kodierte Daten
- CRC-24 Prüfsumme
- Armor Tail Line, die in Abhängigkeit von der ersten Zeile gesetzt wird

[CDF+07] definiert die folgenden Armor Header Lines:

- BEGIN PGP MESSAGE
  Identifiziert verschlüsselte, signierte und komprimierte Nachrichten.
- BEGIN PGP PUBLIC KEY BLOCK
  Identifiziert einen öffentlichen Schlüssel.
- BEGIN PGP PRIVATE KEY BLOCK
  Identifiziert einen privaten Schlüssel.

- BEGIN PGP MESSAGE, PART *X/Y*

  Diese Armor Line wird für mehrteilige Nachrichten[57] verwendet. Sie besagt, dass es sich um den X-ten Teil einer Nachricht, bestehend aus Y Teilen handelt.

- BEGIN PGP MESSAGE, PART *X*

  Entpricht der vorherigen Armor Line, allerdings ist in diesem Fall die genaue Anzahl von Nachrichtenteilen unbekannt.

- BEGIN PGP SIGNATURE

  Identifiziert unter anderem Klartext-Signaturen und abgelöste Signaturen (detached signature).

Die in [CDF+07] beschriebenen Armor Header sind:

- Version

  Dieser Armor Header enthält den Namen und die Version der Anwendung, welche die Nachricht erzeugt hat.

- Comment

  Dieser Header enthält einen frei definierbaren Kommentar im UTF-8 Zeichensatz. Um den Kommentar beim Transport zu schützen sollten allerdings, wie in der kodierten Nachricht auch, nur ASCII-konforme Zeichen benutzt werden.

- MessageID

  Dieser Header enthält eine Zeichenkette der Länge 32, die eine Nachricht eindeutig identifiziert. Dieser Header wird bei mehrteiligen Nachrichten mit unbekannter Anzahl an Teilen eingesetzt, um die separaten Teile der Nachricht zuordnen zu können.

- Hash

  Dieser Armor Header wird nur bei Klartext-Signaturen verwendet. Er enthält eine durch Kommata getrennte Liste der Hashalgorithmen, die beim Signieren verwendet wurden.

- Charset

  Gibt den, für Klartext verwendeten Zeichensatz an. OpenPGP verwendet als Voreinstellung UTF-8.

Die Armor Tail Line markiert das Ende der Nachricht. Sie entspricht vom Aufbau her der Armor Header Line, allerdings wird das Wort `BEGIN` durch `END` ersetzt [CDF+07].

Ein Beispiel für eine Radix-64-konvertierte Nachricht findet sich in [CDF+07]:

[57] Ein Beispiel für eine mehrteilige Nachricht ist beispielsweise eine E-Mail mit mehreren, vom Typ her ggf. unterschiedlichen, Anhängen.

```
-----BEGIN PGP MESSAGE-----
Version: OpenPrivacy 0.99

yDgBO22WxBHv7O8X7O/jygAEzol56iUKiXmV+XmpCtmpqQUKiQrFqclFqUDBovzS
vBSFjNSiVHsuAA==
=njUN
-----END PGP MESSAGE-----
```

4.4 Klartext-Signaturen

Die Verwendung der Radix-64-Kodierung bei signierten Textnachrichten hat den Nachteil, dass der Originaltext nach der Kodierung nicht mehr lesbar ist. OpenPGP ermöglicht für diesen Anwendungsfall die Nutzung von Klartext-Signaturen, bei denen der signierte Text lesbar bleibt und nur die angehängte(n) Signatur(en) selbst Radix-64-kodiert sind. Bei der Erstellung einer Klartext-Signatur wird der urspüngliche Text auf zweierlei Art transformiert. Da in Abhängigkeit vom verwendeten Betriebssystem, unterschiedliche Zeichen einen Zeilenumbruch signalisieren können, werden alle Zeilenumbrüche in das kanonische OpenPGP-Format `<CR><LF>` umgewandelt. Zusätzlich maskiert OpenPGP im Text vorkommende Bindestriche, damit diese bei einer Verarbeitung der Nachricht nicht fälschlicherweise als Armor Header interpretiert werden. Dieser Vorgang wird auch als *dash-escape* bezeichnet. Es muss jedoch beachtet werden, dass für die Hashberechnung der ursprüngliche Klartext, vor den Transformationen, benutzt wird.

Die Struktur einer Klartext-Signatur wird in [CDF+07] wie folgt beschrieben:

- Armor Header Line `-----BEGIN PGP SIGNED MESSAGE-----`
- Ein oder mehrere Hash Armor Header, welche die, für die Signatur verwendeten Hashalgorithmen enthalten.
- Eine Leerzeile
- Der wie oben beschrieben transformierte Klartext.
- Ein oder mehrere Radix-64-kodierte Signaturnachrichten

Kapitel 5

Eclipse Platform

Eclipse ist eine auf Java basierende, universelle Plattform für die Entwicklung von integrierten Entwicklungsumgebungen (IDE[58]), Werkzeugen, sowie verschiedensten Softwarelösungen im Allgemeinen. Ursprünglich von *IBM*[59] im Jahr 2001 ins Leben gerufen, hat Eclipse inzwischen den Wandel zu einem Open-Source-Projekt, bestehend aus mehreren Unterprojekten, vollzogen, an dem eine Vielzahl von (teilweise freiwilligen) Mitarbeitern partizipiert. Es wird unter der *Eclipse Public License* (EPL) V1.0[60] vertrieben, die sicherstellen soll, dass der Eclipse-Code frei verfügbar bleibt, allerdings trotzdem für kommerzielle Anwendungen verwendet werden kann. Dieses Kapitel beschreibt zunächst die Architektur der Eclipse Platform. Im Anschluss wird die Nutzung von Eclipse als Rich Client Platform (RCP) erläutert.

5.1 Eclipse Platform Architektur

Die *Eclipse Platform* bietet primär die Kernfunktionen und -dienste, um eine IDE zu entwickeln [IBM06]. Sie ist modular aufgebaut und besteht aus einer Vielzahl von einzelnen Komponenten, den sogenannten *Plug-ins*. Auf Basis dieser Komponenten können neue Anwendungen erstellt werden, die die Basisfunktionalität der Platform erweitern. Das Spektrum reicht dabei von einfachen integrierten Werkzeugen (Tools), bis hin zu komplexen Applikationen, wie den *Java Development Tools* (JDT), die eine komplette Java Entwicklungsumgebung implementieren oder der Plug-in Entwicklungsumgebung (PDE, *Plug-in Development Environment*) mit deren Hilfe neue Plug-ins erstellt werden können[61]. Ein wichtiger Vorteil der modularen Struktur ist die flexible Integration von Komponenten. Durch die Aggregation bzw. Integration von Plug-ins kann einer Anwendung auf einfache Art und Weise neue Funktionalität hinzugefügt werden. Die Plattform ist zudem sehr generisch konzipiert, dass heißt, obwohl Plug-ins prinzipiell in Java geschrieben werden, ist es durchaus möglich Entwicklungsumgebungen für andere Programmiersprachen oder generell für andere Bereiche der Softwareentwicklung zu imple-

[58] IDE steht für den englischen Begriff Integrated Development Environment.

[59] IBM steht für International Business Machines Corporation und ist eines der größten Unternehmen im Bereich der IT. Siehe auch http://www.ibm.com

[60] Siehe auch http://www.eclipse.org/legal/epl-v10.html

[61] Die Eclipse Platform, JDT und PDE sind Teil des Eclipse Software Development Kits (SDK), welches frei verfügbar unter http://www.eclipse.org/downloads/ heruntergeladen werden kann.

mentieren[62]. Neben Anwendungen aus dem Sektor der Softwareentwicklung, können Rich Client Applikationen erstellt werden, die auf einer Untermenge der Plattformkomponenten aufsetzen, der sogenannten *Rich Client Platform* (RCP). RCP-Anwendungen sind auf keine bestimmte Fachbereichsdomäne beschränkt. So gibt es beispielsweise Software-Lösungen für den medizinischen Sektor, den Bankenbereich oder den Automobilsektor [IBM06]. RCP-Anwendungen werden in Abschnitt 5.4 beschrieben.

Die Komponenten der Eclipse Platform lassen sich in die Bereiche Benutzeroberfläche (User Interface, *UI*) und Kernfunktionalität (*Core*) trennen. Die UI-Komponenten werden von Anwendungen genutzt die eine Benutzeroberfläche benötigen. Die Core-Komponenten implementieren UI-unabhängige Funktionalität und können daher auch in Applikationen ohne Benutzeroberfläche zum Einsatz kommen. Abbildung 5.1 gibt einen Überblick über die Architektur der Eclipse Platform[63].

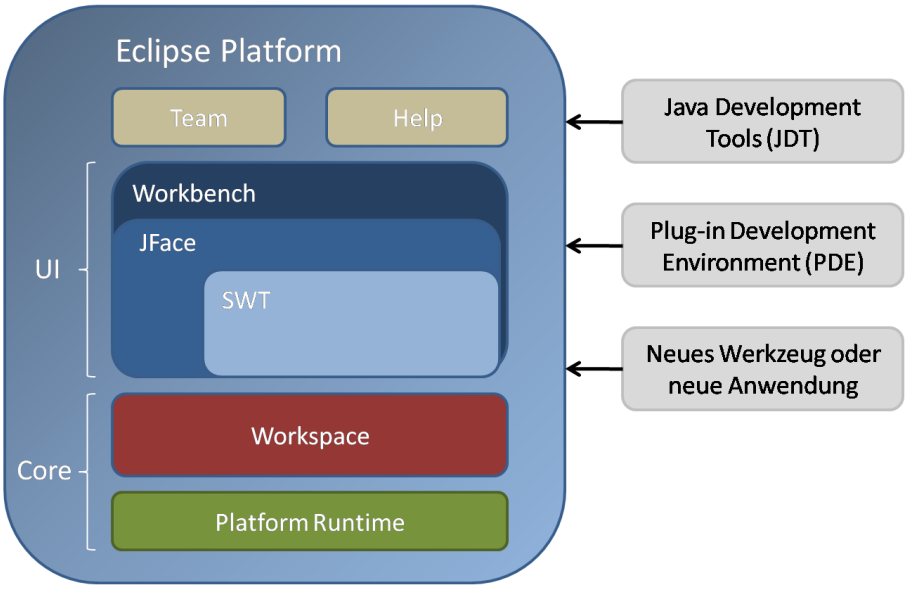

Abbildung 5.1: Eclipse Platform Architektur

Die wichtigsten Komponenten werden im Folgenden beschrieben:

- Platform Runtime

  Die *Platform Runtime* ist die Laufzeitumgebung für Plug-ins. Sie ist für die Verwaltung, den Aufruf und das Beenden der Plug-ins verantwortlich. Vor der Eclipse Version 3.1 nutzte Eclipse ein eigenes Laufzeitsystem. Dabei war es jedoch nicht möglich neue Plug-ins dynamisch hinzuzufügen bzw. zu entfernen. Um dieses Problem zu lösen und eine flexiblere Verwaltung der Plug-ins zu ermöglichen, basieren

[62] Das *C/C++ Development Tooling* Projekt (CDT) bietet beispielsweise eine IDE für C/C++. Siehe http://www.eclipse.org/cdt/

[63] Die Abbildung stellt nur die elementaren Komponenten der Eclipse Platform dar.

neuere Eclipse-Versionen daher auf *Equinox*[64], einer Implementierung der *Open Services Gateway initiative (OSGi) R4 Spezifikation*[65]. [ML05] beschreibt die OSGi Spezifikation als "Rahmen für die Definition, Erstellung und Ausführung von [modularen] Komponenten [...]". Die OSGi Spezifikation nennt diese Komponenten im Gegensatz zu Eclipse *Bundles*. Da ein Plug-in seit Eclipse 3.1 technisch gesehen nichts anderes als ein solches OSGi-Bundle darstellt, können die Begriffe Plug-in und Bundle als synonym betrachtet werden [Del06]. Wenn die Eclipse Platform gestartet wird, identifiziert und analysiert die Runtime vorhandene Plug-ins und erstellt daraus Registrierungsinformationen (*registry*), auf die andere Plug-ins zur Laufzeit über die *Platform API*[66] zugreifen können. Eventuell auftretende Fehler werden dabei protokolliert. Die Registry kann auch zu einem späteren Zeitpunkt verändert werden, so das es möglich ist, Plug-ins zu löschen, zu ersetzen oder neu hinzuzufügen. Um den Speicherverbrauch und die Startdauer der Plattform möglichst gering zu halten, werden die verwalteten Plug-ins nicht direkt beim Start der Plattform aktiviert, sondern erst, wenn der zugrundeliegende Code tatsächlich benötigt wird. Dies wird als *lazy loading* bezeichnet.

- Workspace

  Das Workspace Plug-in ist für die Verwaltung von Anwendungs- oder Werkzeugressourcen wie Ordnern, Dateien etc. zuständig. Diese Ressourcen werden in Form von Projekten organisiert, die sich im Allgemeinen in einem einzigen Workspace-Verzeichnis im Dateisystem befinden. Dieses wird vom Anwender beim Start von Eclipse (einmalig) festlegt[67]. Prinzipiell ist es aber auch möglich Projekte auf verschiedene Dateiverzeichnisse verweisen zu lassen, die sich nicht direkt im Workspace befinden. Der Zugriff auf die Ressourcen innerhalb einer Anwendung ist über ein API möglich. Ein einfaches Beispiel für ein Projekt ist ein Java-Projekt, wie es von den JDT verwendet wird. Es enthält in der Regel den vom Benutzer bearbeiteten Java-Quellcode, kompilierte Dateien, Packagestrukturen in Form von Ordnern, eingebundene Programmbibliotheken etc. Zusätzlich kann einem Projekt eine oder mehrere sogenannte *Naturen* (natures) zugewiesen werden, um automatisch bestimmte Konfigurationseigenschaften mit dem Projekt zu verbinden [IBM06]. Im oben beschriebenen Beispiel könnte das Projekt als *Java Nature* markiert sein.

- Workbench

  Die Workbench stellt die erweiterbare, graphische Benutzeroberfläche von Eclipse dar. Wie in Abbildung 5.1 dargestellt, baut sie auf den beiden Toolkits[68] *Standard Widget Toolkit* (SWT) und *JFace* auf.

[64] Siehe http://www.eclipse.org/equinox/.
[65] Siehe auch http://www.osgi.org/.
[66] API steht für *Application Programming Interface* und definiert die Schnittstelle eines Programms, um anderen Anwendungen den Zugriff auf Dienste dieses Programms zu ermöglichen.
[67] Das Workspace-Verzeichnis kann jederzeit über die Eclipse-Einstellungen angepasst werden.
[68] Ein *Toolkit* (Werkzeugkasten) ist eine Programmbibliothek, die bestimmte Funktionen für die Erstellung von Software zur Verfügung stellt, mit dem Ziel den Entwicklungsaufwand zu verringern.

– Standard Widget Toolkit

Das Standard Widget Toolkit (SWT) ist eine in Java geschriebene Graphikbibliothek[69]. Es stellt eine Sammlung von Basiskomponenten (Widgets) wie Schaltflächen, Auswahllisten, Textfelder, hierarchische Baumdarstellung (Tree) etc., sowie graphische Grundfunktionen zur Verfügung, die für die Entwicklung von Benutzeroberflächen verwendet werden können. Das SWT verwendet für die Darstellung der visuellen Elemente soweit möglich, die native Implementierung des Fenstersystems (window system) des zugrundeliegenden Betriebssystems[70] und emuliert nur Komponenten, die in nativer Form nicht unterstützt werden. Dies hat den Vorteil, dass das SWT schnell ist und auf Eclipse basierende Anwendungen das Look&Feel des Betriebssystems haben, auf dem sie ausgeführt werden. Der Zugriff auf das SWT erfolgt dabei über ein betriebssystemunabhängiges API.

– JFace

Das JFace Toolkit ist eine Bibliothek, die für das Zusammenspiel mit dem SWT entworfen wurde. Während das SWT die graphische Basisfunktionalität implementiert, stellt JFace oft benötigte, komplexere Oberflächenkomponenten (high-level components) wie Dialoge, Assistenten, Benutzerpräferenzen, Fortschrittsanzeige etc. zur Verfügung. Zwei durch JFace zur Verfügung gestellte Eigenschaften sind *Actions* und *Viewer* [IBM06]. Actions erlauben die Entkopplung einer ausgeführten Aktion von den auslösenden Komponenten in der graphischen Benutzeroberfläche. So ist es möglich dieselbe Aktion unterschiedlichen UI Elementen wie z.B. einer Schaltfläche oder einem Menüeintrag zuzuweisen. Dies hat den Vorteil, dass eine bestimmte Aktion nur einmal definiert und dann mehrfach wiederverwendet werden kann. Ein JFace Viewer besteht aus einer SWT Komponente wie z.B. einem Baum (Tree) oder einer Tabelle (Table) und einer Menge von Business- oder Domain-Objekten[71], die in der jeweiligen SWT-Komponente dargestellt werden. Der Viewer kann die Anzeige anhand der Business-Objekte sortieren, filtern, sowie auf Änderungen der Objekte reagieren, um die Anzeige synchron mit den zugrundeliegenden Daten zu halten [Gri02].

Die Workbench wird ausführlicher in Abschnitt 5.2 beschrieben.

- Team

Die Teamkomponente erlaubt die Einbindung von Werkzeugen für die Versionsverwaltung von Projekten im Workspace. Versionierungssysteme sind oft unerläßlich

[69] Eine bekannte Alternative im Javaumfeld ist die von Sun entworfene *Swing*-Bibliothek, die auf dem *Abstract Window Toolkit* (AWT) aufbaut.

[70] Eine Schaltfläche in einem Linuxsystem hat beispielsweise ein anderes visuelles Erscheinungsbild, als in Windows.

[71] Dabei handelt es sich um Entitäten, die für die Abbildung der Geschäftslogik innerhalb einer Anwendung verwendet werden. Im OpenPGP Client wäre ein solches Objekt beispielsweise eine Klasse, die einen transferierbaren Schlüssel repräsentiert.

für die Arbeit in größeren Projekten mit einer Vielzahl von Mitarbeitern, insbesondere wenn mehrere Personen gleichzeitig denselben Quellcode bearbeiten. Die populärsten Vertreter sind *CVS*[72] und *SVN*[73]. Standardmäßig unterstützt Eclipse CVS. Eine Integration von SVN ist beispielsweise über die *Subclipse* Plugins[74] möglich.

- Help

  Der Eclipse Hilfe Mechanismus ermöglicht Werkzeugen und Anwendungen die Einbindung von Dokumentation und Hilfetexten wie z.B. API Beschreibungen oder Benutzerhandbüchern in Form von sogenannten *Online Books*. Der Inhalt eines Online Books setzt sich aus einzelnen *HTML*[75]-Seiten zusammen. Diese werden dem Benutzer in Form eines Baumnavigationsmenüs präsentiert. Die Struktur dieses Baums wird über *XML*[76]-Dateien definiert.

5.2 Eclipse Benutzeroberfläche (Eclipse UI)

Wie schon im vorherigen Abschnitt erläutert, stellt die Workbench die Benutzeroberfläche von Eclipse dar und ist gleichbedeutend mit dem Hauptfenster, welches dem Anwender nach dem Start der Plattform präsentiert wird. Die Darstellung und Strukturierung der visuellen Elemente innerhalb des Fenster, folgt dabei einem, von der Workbench vorgegebenem Muster. Ein Workbench-Fenster besteht im Allgemeinen aus einer Menge von *Sichten* (Views) und *Editoren* (Editors), die den Inhalt einer Anwendung repräsentieren und in Form einer *Perspektive* (Perspective) zusammengefasst werden. Die Perspektive gibt dabei vor, wie die Elemente im Fenster angeordnet und dargestellt werden. Neben Sichten und Editoren enthält die Workbench meist ein Hauptmenü, eine Werkzeugleiste (Toolbar) und ggf. eine Statusleiste. Ein typisches Workbench-Fenster ist in Abbildung 5.2 dargestellt.

Editor
Ein Editor wird für die Bearbeitung eines Objekts, wie z.B. einer Datei, eines graphisches Primitivs in einem visuellen Editor oder einer Datenbanktabelle etc. verwendet. Ein Editor folgt dabei einem bestimmten Lebenszyklus. Das Objekt wird geöffnet, bearbeitet und schließlich gespeichert. Editoren werden in der Regel für die Durchführung einer bestimmten Aufgabe innerhalb einer Anwendung eingesetzt, sie stehen daher im fokalen Mittelpunkt des Workbench-Fensters [ML05] und werden immer innerhalb eines bestimmen Bereichs, der *Editor Area* angezeigt. In Abbildung 5.2 sind beispielsweise mehrere Instanzen eines Editors für die Bearbeitung von Java-Dateien geöffnet. Ein

[72] CVS steht für *Concurrent Versions System* Siehe auch http://www.nongnu.org/cvs/.
[73] SVN steht für *Subversion*. Die Entwickler von SVN hatten das Ziel eine modernere Alternative zu dem älteren CVS zu entwerfen. SVN wird daher oft als Nachfolger von CVS betrachtet. Siehe http://subversion.tigris.org/.
[74] Siehe http://subclipse.tigris.org/
[75] HTML steht für *Hypertext Markup Language*.
[76] XML steht für *Extensible Markup Language*.

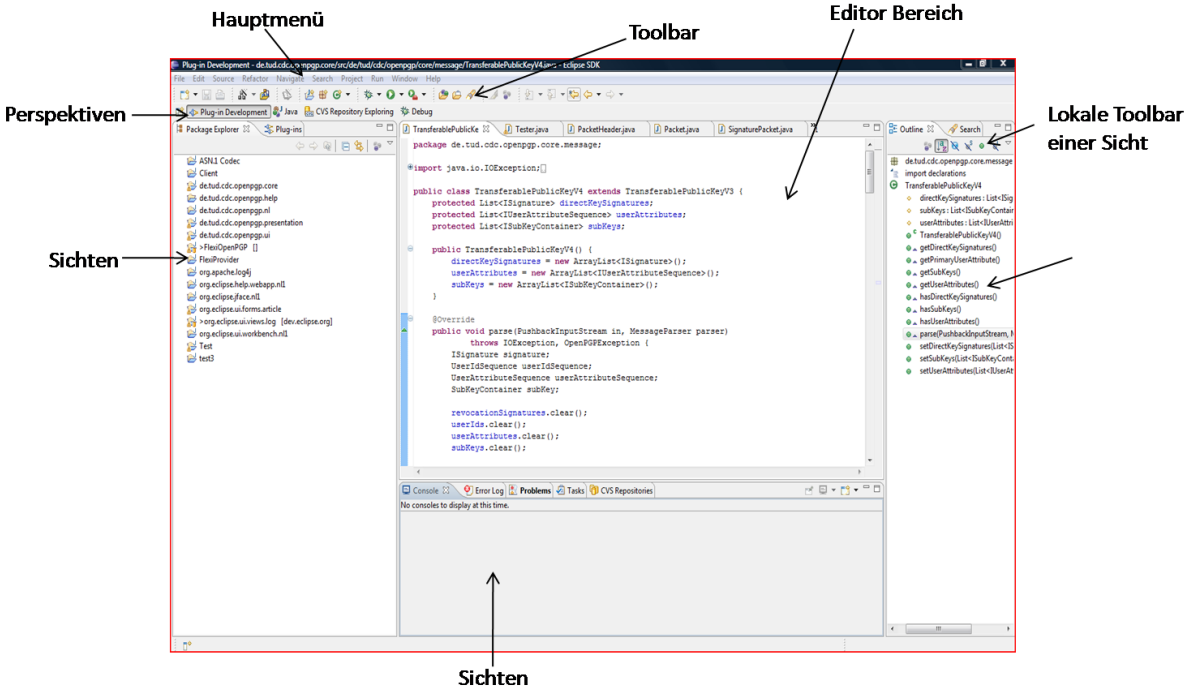

Abbildung 5.2: Eclipse Workbench

aktiver Editor ist innerhalb eines Workbench-Fensters in allen Perspektiven gleichzeitig geöffnet oder geschlossen[77]. Wird er in einer Perspektive geschlossen, wird er somit auch in allen anderen Perspektiven geschlossen. Zudem können Editoren Aktionen zum Hauptmenü und zur Toolbar beisteuern.

Sicht (View)
Eine Sicht stellt meist kontext-sensitive Informationen dar, die in Zusammenhang mit der aktell im Editor ausgeführten Aufgabe stehen. Dies kann z.B. die Darstellung von hierarchischen Informationen, wie die Paketstruktur in einem Java-Projekt oder aber auch die Anzeige der Eigenschaften eines gerade bearbeiteten Objekts o.ä. sein. Die sichtbaren Views werden um die Editor-Area herum im Workbench-Fenster arrangiert. Mehrere Sichten können dabei zu einem sogenannten Stack zusammengefasst werden. Ausserdem ist es möglich eine Sicht aus dem Hauptfenster in ein separates Fenster zu ziehen (detached view). "Sichten haben einen einfacheren Lebenszyklus als Editoren. [Alle durchgeführten] Änderungen innerhalb einer Sicht werden in der Regel sofort gespeichert [...]" [IBM06]. Weiterhin steuert eine Sicht Aktionen nur zu einer lokalen Toolbar bzw. einem lokalen Menü bei.

Perspektive (Perspective)
Eine Perspektive ist für das Layout der Sichten und des Editorbereichs verantwortlich. Alle Komponenten werden dabei innerhalb des Worbench-Fenster relativ zueinander

[77] Es ist allerdings möglich innerhalb einer Perspektive die Sichtbarkeit der EditorArea zu modifizieren.

platziert. Zudem kontrolliert sie die initiale Sichtbarkeit von Views und Aktionen. Pro Workbench-Fenster können mehrere Perspektiven definiert sein, die Sichten und Editoren für eine bestimmte Aufgabe arrangieren (z.B. eine Perspektive für Plug-in Entwicklung, CVS, Debugging etc.). Es kann allerdings immer nur genau eine Perspektive aktiv sein. Zur Laufzeit kann zwischen den verschiedenen Perspektiven gewechselt werden.

5.3 Plug-in Architektur

Wie bereits in Abschnitt 5.1 beschrieben, stellen Plug-ins die funktionale Basiseinheit in Eclipse dar. Für gewöhnlich wird ein Plug-in in Java geschrieben, es gibt jedoch auch Plug-ins die ihre Funktionalität komplett ohne Quellcode erbringen. Ein Beispiel hierfür ist ein Plug-in, das die XML- und HTML Dateien für ein Onlinebook enthält, welches über die Eclipse Hilfe angezeigt werden kann. Ein Plug-in kann auch aus mehreren *Fragment*-Plug-ins bestehen. Dies ist insbesondere bei der Internationalisierung einer Anwendung von Vorteil. Der Java-Code kann beispielsweise im sogenannten *Host-* oder *Target-Plug-in* abgelegt werden, während die Fragmente die sprachabhängigen Übersetzungen enthalten. Zur Laufzeit werden Host- und Fragment-Plug-ins dann zusammengeführt. Plug-ins werden als Java Archive[78] (JAR) gespeichert. Innerhalb der Plug-in-Struktur befinden sich zwei Dateien, die von zentraler Bedeutung sind, `/META-INF/MANIFEST.MF` und `plugin.xml`.

/META-INF/MANIFEST.MF[79]
Die Datei `MANIFEST.MF` ist ein OSGi-Bundle Manifest (siehe hierzu die Erläuterung der Platform Runtime in Abschnitt 5.1). Sie befindet sich standardmäßig im Ordner `META-INF` innerhalb der JAR-Struktur und enthält Metainformationen über das Plug-in. Hier zu zählen u.a. ein Bezeichner (*Bundle-SymbolicName*), die Version des Plug-ins, sowie ein beschreibender Name. Der Bezeichner und die Plug-in Version werden kombiniert, um das Plug-in eindeutig zu identifizieren. Der Eintrag *Bundle-Activator* gibt die mit dem Plug-in assoziierte Klasse an, die verwendet wird, um das Plug-in zu starten und zu stoppen. Dabei handelt es sich um die erste Klasse die bei der Plug-in-Ausführung aufgerufen wird. Die Activator-Klasse ist in der Regel eines Subklasse von `org.eclipse.core.runtime.Plugin`, sofern das Plug-in keine UI Funktionalität erweitert bzw. `org.eclipse.ui.plugin.AbstractUIPlugin` im anderen Fall. Alle Klassen innerhalb eines Plug-ins sind zunächst gekapselt und nicht für andere Plug-ins sicht-

[78] Java Archives wurden ursprünglich für den kompakten Download von Java Applets eingeführt, um Bandbreite zu sparen und die notwendigen Daten in einem Paket zu laden. Sie werden aber oft auch für das Deployment von Java Bibliotheken oder Paketen verwendet. Ein JAR ist ein ZIP komprimiertes Archiv, das mit Metadaten über den Inhalt des Archivs angereichert werden kann. Die Metadaten stehen in einer sogenannten Manifest-Datei `MANIFEST.MF`, die sich im Verzeichnis `META-INF` befindet. Siehe http://java.sun.com/javase/6/docs/technotes/guides/jar/jarGuide.html und http://java.sun.com/javase/6/docs/technotes/guides/jar/jar.html.

[79] Die hier beschriebene Darstellung bezieht sich auf Eclipse ab Version 3.1. In vorherigen Versionen wurden die Plug-in Dateien in einem Plug-in Verzeichnis (Plug-in Directory) zusammengefasst. Zudem enthielt die plugin.xml zusätzlich die Metainformationen aus der Datei `MANIFEST.MF`.

bar. Jedes Plug-in reguliert den Zugriff auf die eigenen definierten Klassenpakete im Manifest und gibt zudem an, welche anderen Plug-ins bei der Ausführung benötigt werden. Diese Informationen stehen in den Sektionen *Export-Package* und *Require-Bundle*. Das Manifest des Plug-ins, das die graphische Benutzeroberfläche des OpenPGP Clients implementiert, sieht beispielsweise wie folgt aus:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: OpenPGP Client Plug-in
Bundle-SymbolicName: de.tud.cdc.openpgp.ui;singleton:=true
Bundle-Version: 1.0.0
Require-Bundle: org.eclipse.core.runtime,
org.eclipse.ui,
de.tud.cdc.openpgp.core,
org.eclipse.ui.forms,
org.eclipse.ui.intro,
org.eclipse.ui.views.log,
org.eclipse.ui.views
Eclipse-LazyStart: true
Bundle-Vendor: TU Darmstadt
Bundle-Activator: de.tud.cdc.openpgp.UIActivator
Bundle-Localization: plugin
Export-Package: de.tud.cdc.openpgp.ui
```

plugin.xml

Jedes Plug-in kann sogenannte *Extension Points* (Erweiterungspunkte) definieren, die es anderen Plug-ins ermöglichen, bestehende Funktionalität dieses Plug-ins zu erweitern. Dieser Mechanismus stellt die Basis für die Integration von Werkzeugen und Anwendungen in Eclipse dar. So stellt beispielsweise das Plug-in `org.eclipse.ui` einen Extension Point für die in Abschnitt 5.2 beschriebenen Views zur Verfügung. Wenn eine Anwendung Informationen in einem View darstellen möchte, erweitert sie den Extension Point `org.eclipse.ui.views`[80] und erstellt eine Klasse, die von der abstrakten Basisklasse `ViewPart` erbt. Die `plugin.xml` Datei enthält alle von einem Plug-in definierten Extension Points, sowie alle von diesem Plug-in erweiterten Extensions Points. Ein Auschnitt aus der entsprechenden Datei des OpenPGP Client UI-Plug-ins zeigt die Deklaration einer View-Extension:

```
<extension point="org.eclipse.ui.views">
    <view
      id="de.tud.cdc.openpgp.ui.views.keys"
      name="%keyViewTitle"
      class="de.tud.cdc.openpgp.ui.views.KeysView"
```

[80] Dies wird als *Contribution* bezeichnet, da dem Plug-in, das den Extension-Point zur Verfügung stellt, neue Funktionalität hinzugefügt wird.

```
      icon="icons/keys_24x24.png''
    />
</extension>
```

Wie in dem Ausschnitt ersichtlich, kann ein Extension-Point für die Erweiterung notwendige Attribute vorgeben. In dem Beispiel werden neben der Kennung und der implementierenden Klasse auch der, in der Titelleiste des Views angezeigte Name, sowie das dort dargestellte Icon festgelegt.

Die Informationen aus `MANIFEST.MF` und `plugin.xml` werden von der Runtime genutzt, um die Registrierung zu erstellen. Für weitere Informationen wird auf [IBM06], [ML05] und [Del06] verwiesen.

5.4 Eclipse Rich Client Platform

Während die Eclipse Platform primär für die Entwicklung von IDEs konzipiert wurde, lassen sich mit der Rich Client Platform (RCP) Anwendungen für beliebige Bereiche erstellen. Der Vorteil für Entwickler dabei ist, dass Grundfunktionalitäten bereits durch die, von Eclipse selbst zur Verfügung gestellten Komponenten vorhanden sind und mehr der Entwurf und die Umsetzung der Geschäftslogik der Applikation im Vordergrund stehen. Allerdings ist der Eclipse-Framework relativ komplex und es bedarf einer gewissen Einarbeitungszeit, um die Zusammenhänge und Strukturen des Systems nachzuvollziehen und erfolgreich Anwendungen zu entwickeln.

Wie in Abschnitt 5.1 erwähnt, stellt die RCP eine Untermenge der Eclipse Platform dar. Damit in der Anwendung die Eclipse Laufzeitumgebung und eine Benutzeroberfläche zur Verfügung stehen, setzt eine RCP Anwendung mit minimaler Konfiguration auf den Plattform Komponenten `org.eclipse.core.runtime` und `org.eclipse.ui` auf. Diese Konfiguration kann natürlich um zusätzliche Eclipse Komponenten (Help, Update etc.) und auch durch eigene Plug-ins erweitert werden. Jede RCP Anwendung besteht aus einem Plug-in und definiert eine Klasse, die als Startpunkt, ähnlich der *main()*-Methode in klassischen Programmen, fungiert. Die Klasse implementiert die Schnittstelle `org.eclipse.equinox.app.IApplication` und wird über den *Application* Extension Point innerhalb der `plugin.xml` deklariert. Das Plug-in, dass die Application-Klasse beinhaltet, wird initial von der Runtime gestartet. Der Name der auszuführenden Anwendung wird dabei beim Start der Platform per Kommandozeile übergeben [IBM06]. Die Anwendung behält dann bis zu Ihrer Beendigung die gesamte Kontrolle über Eclipse. Demzufolge kann zur Laufzeit auch nur genau eine Applikation aktiv sein. Ein *Produkt* (product) erweitert eine Applikation um sogenannte *Marken*-Informationen (branding information), die dazu genutzt werden können, das visuelle Erscheinungsbild einer Anwendung an ein bestimmtes, nach außen transportiertes Firmendesign (Corporate Design) anzupassen. Dazu gehören z.B. ein angepasster Splash-Screen, verwendete Icons, About-Dialog etc. Ein Produkt wird über den *Product* Extension Point deklariert und re-

ferenziert die zu erweiterende Applikation über den eindeutigen Application Bezeichner (Application ID). Die Definition eines Produkts ist optional.

Kapitel 6

Design und Implementierung des OpenPGP Clients

Im folgenden Kapitel werden das grundlegende Design, sowie die konkrete Realisierung des OpenPGP Clients beschrieben. Zu Beginn wird kurz die Motivation für die Verwendung der Eclipse RCP Platform, sowie die Entwicklungsumgebung erläutert. Es folgt ein kurzer Überblick über die, im OpenPGP Client verwendeten Plug-ins. Danach wird die bestehende OpenPGP Bibliothek analysiert und Vorschläge für eine Verbesserung des Quellcodes erarbeitet. Im Anschluss wird die Architektur der neuen OpenPGP Bibliothek dargestellt. Nach einer Beschreibung der Benutzeroberfläche mit den einzelnen Funktionsbereichen, werden abschließend die Themen Benutzerhandbuch, Fehlerprotokollierung und Internationalisierung diskutiert.

6.1 Motivation für die Verwendung der Eclipse RCP Platform

Der OpenPGP Client wird als Eclipse RCP Anwendung konzipiert. Für die Wahl der Eclipse Platform sprachen verschiedene Gesichtspunkte. Wie in Kapitel 5 beschrieben, stellt Eclipse dem Entwickler eine große Sammlung verschiedener Funktionen standardmäßig zur Verfügung. So können komplexe Anwendungsbausteine wie Assistenten (Wizards), Benutzereinstellungen (User Preferences), Hilfemodul, Updateservice etc. mit relativ wenig Aufwand in eine eigene Applikation integriert werden. Zudem lassen sich mit SWT und JFace moderne Benutzeroberflächen gestalten, die (sofern möglich) das Look&Feel des verwendeten Betriebssystems nutzen. Ein weiterer wichtiger Entscheidungspunkt ist das modulare Design von Eclipse. Durch Hinzufügen von neuen Plug-ins kann der OpenPGP Client flexibel um zusätzliche Funktionen erweitert werden. Eine mögliche Alternative zur Eclipse Platform ist die *NetBeans Platform*[81] von Sun Microsystems[82], mit der ebenfalls Rich Client Anwendungen erstellt werden können[83]. Wie Eclipse ist auch Netbeans komponentenbasiert, baut für die graphische Darstellung al-

[81] Siehe http://www.netbeans.org/ und insbesondere http://platform.netbeans.org/.
[82] Siehe http://de.sun.com/
[83] Für einen Vergleich der beiden Plattformen siehe beispielsweise http://blogs.sun.com/geertjan/entry/eclipse_platform_vs_netbeans_platform.

lerdings auf Swing[84] auf. Da jedoch einige parallele Projekte am Institut bereits Eclipse nutzen, erscheint die aktuell gewählte Lösung, insbesondere unter dem Gesichtspunkt einer späteren Integration, als zukunftsorientierter.

6.2 Entwicklungsumgebung

Für die Entwicklung des OpenPGP Client wird die zur Zeit aktuelle Eclipse Version 3.3.1 verwendet. Als Java Laufzeitumgebung (Java Runtime Environment, JRE) kommt Java 6 Update 2 zum Einsatz. Die Java Laufzeitumgebung wird mit der Anwendung ausgeliefert, dass heisst auf dem Zielsystem muss kein Java installiert sein.

6.3 Übersicht OpenPGP Client Plug-ins

Der OpenPGP Client besteht aus einer Reihe von Plug-ins. Die wichtigsten dieser Plug-ins werden hier in Kurzform vorgestellt und im weiteren Verlauf des Kapitels detailiert erläutert[85].

- `de.tud.cdc.openpgp.core`

  Das Core-Plug-in enthält die aktualisierte Version der OpenPGP Bibliothek, sowie die Datenbank, in der die OpenPGP Schlüssel gespeichert werden (siehe Abschnitt 6.5).

- `de.tud.cdc.openpgp.ui`

  Das UI-Plug-in implementiert die eigentliche Anwendung (Application Klasse und Produkt) sowie die Benutzeroberfläche des OpenPGP Clients (siehe Abschnitt 6.6).

- `de.tud.cdc.openpgp.presentation`

  Implementiert die Präsentation des OpenPGP Clients, die verwendet wird, um das Look&Feel der Anwendung anzupassen (siehe Abschnitt 6.6.2.2).

- `de.tud.cdc.openpgp.help`

  Enthält eine einfache Vorlage für eine Benutzerhandbuch, welches innerhalb des Eclipse Hilfesystems angezeigt werden kann.

- `org.eclipse.ui.views.log`

  Dieses Plug-in implementiert den Standard Eclipse Error Log View, der für die Anzeige von aufgetretenen Fehlern, Warnungen etc. verwendet wird (siehe Abschnitt 6.9).

[84] Siehe http://java.sun.com/javase/technologies/desktop/

[85] Die Liste aller genutzten Plug-ins findet sich nach Start des Programms im About-Dialog unter Plug-in Details.

- `de.tud.cdc.openpgp.ui.nl`

  Dieses Plug-in enthält die lokalisierten Übersetzungen der im UI-Plug-in verwendeten Texte.

- `de.tud.cdc.openpgp.core.nl`

  Dieses Plug-in enthält die lokalisierten Übersetzungen der im Core-Plug-in verwendeten Texte.

- `org.eclipse.ui.workbench.nl1`

  Dieses Plug-in enthält die lokalisierten Übersetzungen der im Workbench-Plug-in verwendeten Texte.

- `org.eclipse.ui.jface.nl1`

  Dieses Plug-in enthält die lokalisierten Übersetzungen der im JFace-Plug-in verwendeten Texte.

- `org.eclipse.help.webapp.nl1`

  Dieses Plug-in enthält die lokalisierten Übersetzungen der im Hilfedialog verwendeten Texte.

6.4 Analyse der bestehenden OpenPGP-Bibliothek

Ursprünglich sollte die OpenPGP Bibliothek in der bestehenden Form als Basis für den OpenPGP Client verwendet werden. Der primär angestrebte Fokus dieser Arbeit lag auf dem Design einer flexiblen Oberfläche, die den Zugriff auf die, durch die Bibliothek zur Verfügung gestellte Funktionalität, ermöglichen sollte. Leider zeigten sich bei genauerer Analyse des Praktikumscodes zahlreiche Probleme, die letztlich zu weitreichenden Änderungen am ursprünglichen Code geführt haben. In diesem Abschnitt werden die wichtigsten Punkte die zu dieser Entscheidung geführt haben erläutert.

Nachrichtenverarbeitung
OpenPGP Nachrichten werden von der Bibliothek als *byte*-Array verarbeitet. Eine oder mehrere Nachrichten werden dabei aus einer Datei eingelesen und dann im Speicher verarbeitet. Diese Lösung ist relativ einfach umzusetzen, hat aber den Nachteil, dass sehr große Nachrichten (GnuPG beispielsweise verarbeitet problemlos Dateien größer als 500MB) je nach vorhandenem Speicher im Rechner des Anwenders gar nicht, oder nur sehr inperformant analysiert und verarbeitet werden können. Zudem werden beim Einlesen von Datenpaketen, insbesondere auch bei Verwendung von partieller Kodierung, redundante Kopien der Nachrichtendaten erstellt, was zu einer weiteren Erhöhung der Speicherlast führt. Um diese Probleme zu umgehen stehen prinzipiell zwei Möglichkeiten zur Verfügung. Zum einen können in Java sogenannte *MemoryMapped-Files* verwendet werden. Die relevante Klasse hierbei ist `FileChannel`[86] aus dem Paket `java.nio`. Sie

[86]Siehe http://java.sun.com/javase/6/docs/api/java/nio/channels/FileChannel.html

verfügt über die `map`-Methode, bei der ein Teil einer Datei im Speicher abgelegt wird. Der Zugriff erfolgt dann ähnlich wie auf ein *byte*-Array[87] Der Vorteil dieser Methode ist, dass nicht mehr die komplette Datei eingelesen werden muss, andererseits weist die Java API Beschreibung darauf hin, dass dieser Mechanismus zum Großteil vom zugrundeliegenden Betriebssystem abhängt und sich ein Performancegewinn nur bei relativ großen Dateien erzielen lässt[88]. Die andere Möglichkeit ist eine streambasierte Verarbeitung der Nachrichten, die auf den Klassen (und Subklassen) des `java.io`[89] Pakets aufsetzt. Die Daten werden dabei in kleinen Blöcken als Datenstrom eingelesen bzw. geschrieben. Der Zugriff erfolgt über die verschiedenen `read()`- und `write()`-Methoden der Klassen `InputStream` und `OutputStream` und davon abgeleiteten Subklassen. Der Vorteil dieser Alternative ist ebenfalls eine sehr geringe Speicherauslastung, sowie die mögliche Nutzung von bereits von Java zur Verfügung gestellten Filterklassen (um zum Beispiel in einem Stream Daten direkt zu verschlüsseln, zu komprimieren etc.). Diese Variante wird in der neuen Bibliothek verwendet und im Abschnitt 6.5.2 ausführlich diskutiert.

Paketverarbeitung
Pakete mit unbestimmter Größe werden in einigen Fällen fehlerhaft verarbeitet. Es wird implizit davon ausgegangen, dass Pakete diesen Types sich immer bis zum Ende einer Datei erstrecken. Dies kann bei Schachtelung von mehreren Nachrichten dazu führen, dass die Daten nicht korrekt interpretiert werden. Da die Größe von Paketen durch die Verarbeitung im Speicher immmer im Voraus bekannt ist, wurde zudem nur das Einlesen von partiell kodierten Paketen implementiert, das Schreiben solcher Pakete ist nicht möglich.

Algorithmen
Die Bibliothek verwendet keine repräsentative Darstellung in Form von Klassen, für die von OpenPGP genutzten kryptographischen Algorithmen. Innerhalb der Pakete erscheinen daher oft typische `if-then-else` Abfragen, die auf einen konkreten Algorithmus prüfen. Dies soll mit der Verwendung von Interfaces und dem *Strategy*-Pattern[90] umgangen werden. Algorithmeneigenschaften wie eindeutiger Bezeichner, Schlüssel-, und Blockgröße etc. sind über mehrere Klassen verteilt und werden nicht zentral verwaltet. Eine neue Klasse, die für die Administration der Algorithmen verantwortlich ist, soll dieses Problem beheben. Einige Algorithmen wie der Kompressionsalgorithmus *BZip2* oder das Public-Key-Verfahren *ElGamal* werden nicht unterstützt.

Verschlüsselung
OpenPGP verwendet für die symmetrische Verschlüsselung eine spezielle Variante des

[87] Konkret erfolgt der Zugriff über einen `MappedByteBuffer`, eine Subklasse von `ByteBuffer`. Siehe hierzu http://java.sun.com/javase/6/docs/api/java/nio/MappedByteBuffer.html.
[88] Siehe hierzu die Erläuterung der `map`-Methode in der API-Beschreibung der `FileChannel`-Klasse unter http://java.sun.com/javase/6/docs/api/java/nio/channels/FileChannel.html.
[89] Siehe http://java.sun.com/javase/6/docs/api/java/io/package-summary.html
[90] Siehe [FFS04] oder [GHJ04].

Cipher Feedback Mode[91] (CFB). Der CFB ist ein bestimmter Betriebsmodus von Blockchiffren. Blockchiffren sind "[...] Verschlüsselungsverfahren, die [Daten-]Blöcke fester Länge auf [verschlüsselte] Blöcke derselben Länge abbilden" [Buc03]. Beim CFB gehen bereits verschlüsselte Daten einer Nachricht in die weitere Verschlüsselung von Daten dieser Nachricht mit ein (es besteht also eine Abhängigkeit bei der Verarbeitungsreihenfolge). Der CFB verwendet hierzu ein Schieberegister, in dem Daten für die Weiterverarbeitung gespeichert werden[92]. Der CFB-Modus von OpenPGP schreibt vor, dass die Größe des Schieberegisters der Blockgröße des verwendeten Algorithmus entspricht. Dies wurde im Originalcode nur bedingt berücksichtigt, es kann dort zu fehlerhafter Initialisierung des Verfahrens kommen. Gleichermaßen wird die Größe des Initialisierungsvektors (IV)[93] des Verfahrens (die ebenfalls der Blockgröße entsprechen muss) nicht immer korrekt gesetzt. OpenPGP schreibt bei der Verschlüsselung von symmetrischen Datenpaketen zudem einen *Resynchronisationsschritt* vor, bei dem der Inhalt des Schieberegisters vor der Ver-/Entschlüsselung der eigentlichen Daten angepasst wird. Dieser Mechanismus wurde nicht implementiert. Weiterhin werden dem Klartext vor der Verschlüsselung laut OpenPGP Spezifikation zufällig generierte Daten vorangestellt (random prefix data). Bei der Entschlüsselung wird dies in der aktuellen Version nicht berücksichtigt. Stattdessen werden, neben dem reinen Klartext, auch immer die zufällig generierten Prefixdaten mit zurückgegeben. Die Prefixdaten erlauben zudem eine schnelle Überprüfung (Quick-Check) der Gültigkeit des Sitzungsschlüssels. Der Quick-Check wurde fehlerhaft umgesetzt. Wie in Kapitel 4 beschrieben, ist es ausserdem möglich, dass der Sitzungsschlüssel einer Nachricht durch ein oder mehrere *Symmetric-Key Encrypted Session Key Packets* oder *Public-Key Encrypted Session Key Packets* geschützt werden kann. Die Bibliothek erlaubt allerdings nur die exklusive Verwendung der Session Key Pakete, dass heisst man schützt den Sitzungschlüssel entweder symmetrisch mit ein oder mehreren Passphrasen oder mit öffentlichen Schlüsseln. Die Verwendung beider Methoden ist nicht möglich. Die angesprochenen Punkte legen nahe, dass die Ver-/Entschlüsselung von Daten nicht mit konkreten OpenPGP Nachrichten getestet wurde.

Kompression
Bei der Kompression von Daten werden, je nach verwendetem Verfahren, temporäre Dateien angelegt. Dies wird durch die Umstellung der Nachrichtenverarbeitung (s.o.) umgangen. Das Verfahren BZip2 fehlt in der Bibliothek und wird in der neuen Version ergänzt.

Signieren/Verifizieren
Wie in Kapitel 4 dargestellt, unterscheidet OpenPGP zwischen verschiedenen Signaturtypen. Die Berechnung einer konkreten Signatur variiert dabei, in Abhängigkeit vom Signaturtyp. Im bestehenden Code geht diese Berechnung nur beim Signieren von Daten

[91] Siehe [CDF+07] Sektion 5.7., 5.13. und 13.9.
[92] Für eine ausführliche Beschreibung von Blockchiffren und dem CFB siehe [Buc03] Kapitel 4.6 und 4.8.3.
[93] Der IV ist ein Block mit meist zufällig generierten Daten, der als Initialisierungswert für bestimmte Modi von Blockchiffren wie z.B. dem CFB verwendet wird.

mit ein, die Verifikation ist nur für Signaturen von Text- und Binärdaten implementiert. Das Feld Signaturtyp innerhalb eines Signaturpakets wird weitestgehend ignoriert und bei Neuerzeugung von Signaturen nicht korrekt gesetzt. Zudem fehlt die Umsetzung einiger Signaturtypen wie Widerruf eines Unterschlüssels, direkte Schlüsselsignatur etc. Die Bibliothek wird um die fehlende Funktionalität erweitert.

Schlüssel
Die Bibliothek verarbeitet transferierbare Schlüssel in der Version 4. Schlüssel der Version 3 werden nur bedingt bis gar nicht unterstützt. In der neuen Version können beide Schlüsselversionen verwendet werden. Einige wichtige Eigenschaften von transferierbaren Schlüsseln stehen nicht zur Verfügung, so fehlt beispielsweise die Berechnung von primären Benutzereigenschaften (primäre Benutzerkennung bzw. Benutzerattribut), die Größe des verwendeten Schlüssels in Bit, Berechnung des Ablaufdatums anhand der Selbstzertifikate etc. Beim Einlesen der Schlüssel wird keine Überprüfung der Signaturen oder des Ablaufsdatum durchgeführt. Weiterhin ist es nicht möglich einen neuen Schlüssel zu generieren, stattdessen kann ein Schlüsselobjekt lediglich mit bereits vorhandenen Parameterwerten befüllt werden.

Allgemeine Probleme
Neben den bereits dargestellten Problemen wirkt die Bibliothek insgesamt unfertig, so machen einige Programmierfehler (z.B. `null`-Pointer Exceptions beim Signieren/-Verifizieren von Daten) die zur Verfügung stehende Funktionalität nur eingeschränkt nutzbar. An vielen Stellen finden sich `if-then-else` Anweisungen, die zwischen verschiedenen konkreten Objekttypen unterscheiden. Dies soll durch die Verwendung von Design-Patterns und dem Einführen von abstrakten Klassen bzw. Interfaces umgangen werden. Zudem sind die Verantwortlichkeiten der jeweiligen Klassen nicht immer eindeutig definiert. Die Klasse `OpenPGPHelper` beispielsweise ist eine Allzweckklasse, die eine Vielzahl von Aufgaben erfüllt. Sinnvoller wäre es die Funktionalität auf verschiedene Klassen aufzuteilen.

Als Fazit lässt sich zusammenfassen, dass die oben aufgeführten Punkte eine Neuimplementierung der OpenPGP Bibliothek rechtfertigen. Die beschriebenen Probleme sollen dabei behoben bzw. fehlende Funktionalität ergänzt werden. Eine vollständige Implementierung des OpenPGP Standards ist aufgrund der zeitlichen Begrenzung und der zusätzlichen Entwicklung der Benutzeroberfläche nicht möglich. Daher liegt der Schwerpunkt der Neuentwicklung auf der Erzeugung einer möglichst stabilen Codebasis, die dann in Folgeprojekten zu einer vollständigen Umsetzung erweitert werden kann.

6.5 Neuentwicklung der OpenPGP Bibliothek

Die neue OpenPGP Bibliothek mit allen zugehörigen Klassen befindet sich im Plug-in `de.tud.cdc.openpgp.core` (siehe Abbildung 6.1) und umfasst, neben der Teilumsetzung des OpenPGP Standards, eine Datenbank zur Speicherung von asymmetrischen

Schlüsseln, Methoden für die Erzeugung, den Import und Export von Schlüsseln, eine Implementation des Web of Trust sowie zahlreiche Hilfsklassen. Sie basiert, wie auch die Vorgängerversion, auf den Klassen der *Java Cryptography Architecture*[94] (JCA). Die JCA ist Teil des Java Development Kits (JDK) und stellt die kryptographischen Grundfunktionen wie Verschlüsselung, Signieren und Hashberechnung sowie die Möglichkeit der Schlüsselerzeugung zur Verfügung. Hierfür sind insbesondere die Klassen aus den Paketen `javax.crypto` und `java.security` des JDK von Bedeutung.

Da die Standardimplementation der JCA nicht alle im OpenPGP Format beschriebenen Algorithmen unterstützt, wurde zusätzlich das, an der TU Darmstadt entwickelte *FlexiProvider Toolkit*[95] in die OpenPGP Bibliothek integriert. Im Folgenden werden die

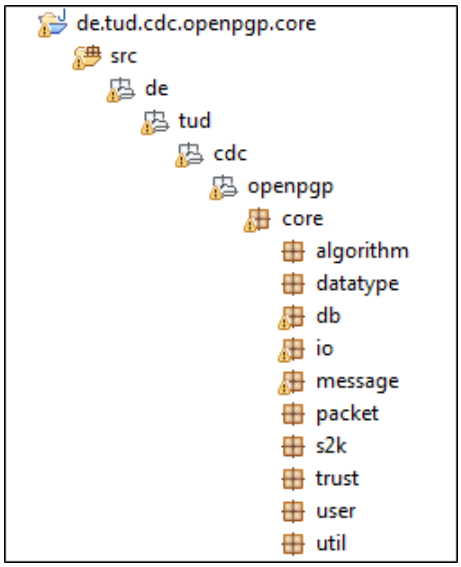

Abbildung 6.1: Plug-in Struktur der OpenPGP Bibliothek

verschiedenen Bereiche der Bibliothek ausführlich dargestellt.

6.5.1 Verwaltung der Algorithmen

Wie in Kapitel 4 beschrieben, definiert OpenPGP eine Vielzahl von symmetrischen und asymmetrischen Verschlüsselungs- sowie Hash- und Kompressionsalgorithmen, die für die kryptographischen Grundfunktionen verwendet werden können[96]. Die verschiedenen OpenPGP Pakete referenzieren einen solchen Algorithmus über einen, im Standard

[94] Siehe auch http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.
[95] Das FlexiProvider Toolkit ist eine auf der JCA aufbauende Open-Source Lösung, die eigenständige Umsetzungen verschiedener kryptographischer Algorithmen implementiert und die JCA Referenzimplementaion um zusätzliche Funktionalität erweitert. Siehe http://www.cdc.informatik.tu-darmstadt.de/flexiprovider/
[96] Siehe [CDF+07] Kapitel 9.1 bis 9.4.

eindeutig festgelegten Bezeichner (ID). So steht die ID *9* bei symmetrischen Verschlüsselungsalgorithmen beispielsweise für AES mit einer Schlüsselgröße von 256 Bit. Die in der Einleitung erwähnten Basisklassen aus den Paketen `javax.crypto` und `java.security` werden hingegen mit einem vordefinierten Algorithmusnamen (z.B. Blowfish, RSA, SHA-1 etc.) und ggf. mit zusätzlichen Parametern wie z.B. der zu verwendeten Schlüsselgröße initialisiert, dass heißt es muss eine Abbildung der OpenPGP Algorithmen ID auf einen JCA kompatiblen Algorithmennamen mit den zugehörigen Algorithmeneigenschaften erfolgen. Zu diesem Zweck wurden der Bibliothek neue Klassen hinzugefügt, die die verschiedenen Algorithmen-Typen repräsentieren. Sie liegen im Paket `de.tud.cdc.openpgp.core.algorithm`.

Um eine dynamische Zuweisung der Algorithmen innerhalb der einzelnen Pakete zu erlauben, folgt die Implementierung der Algorithmenobjekte dem Strategy-Pattern[97]. Eine bestimmter Algorithmus mit allen zugehörigen Eigenschaften wird durch eine Klasse vom Typ `IAlgorithm` oder einer Subklasse davon repräsentiert. `IAlgorithm` ist die Basisschnittstelle aller Algorithmen. Sie definiert die folgenden Grundoperationen:

- `String getDescription()`

  Liefert die textuelle Beschreibung des JCA Algorithmusnamens.

- `int getAlgorithm()`

  Liefert die OpenPGP ID des Algorithmus.

Pro Algorithmustyp gibt es ein weiteres Interface, dass entweder zusätzliche Operationen, die für den jeweiligen Typ relevant sind hinzufügt oder für ein einfaches Typecasting verwendet werden kann (z.B. `ICompressionAlgorithm`). Die komplette Hierarchie von `IAlgorithm` mit den in den Interfaces definierten Methoden ist in Abbildung 6.2 dargestellt. Erwähnenswert hierbei sind insbesondere die Methoden `decodeJCA*`, `encodeJCA*` und `parse*` im Interface `IPublicKeyAlgorithm`, die in den implementierenden Klassen für die Verarbeitung und Konvertierung der Parameter von asymmetrischen Verfahren verantwortlich sind. Die `parse*`-Methoden werden verwendet, um Schlüsselparameter in Abhängigkeit vom Algorithmentyp, für das jeweilige OpenPGP Paket (z.B. ein PublicKey-Paket) transparent einzulesen. Die einzelnen Pakete wissen dabei nichts über die konkreten Parameter eines asymmetrischen Algorithmus, sondern delegieren die Verarbeitung an die entsprechenden Algorithmenobjekte weiter. Die `decode*`- und `encode*`-Methoden werden benutzt, um Schlüsselparameter vom OpenPGP-Format in das von den JCA-Klassen erwartete Parameterformat zu konvertieren und umgekehrt. Sie erfüllen somit eine "Adapterfunktion". Dies ist z.B. notwendig, da die OpenPGP Pakete große Ganzzahlen als *Multiprecision Integer* (MPI)[98] speichern, während die JCA Objekte, die Parameter als `java.math.BigInteger` erwarten.

[97] Siehe [FFS04] oder [GHJ04].

[98] Ein MPI ist eine vorzeichenlose Ganzzahl, der ein 2-Oktet Wert vorangestellt wird, in dem die kodierte Länge der Ganzzahl in Bit enthalten ist. Siehe [CDF+07] Sektion 3.2.

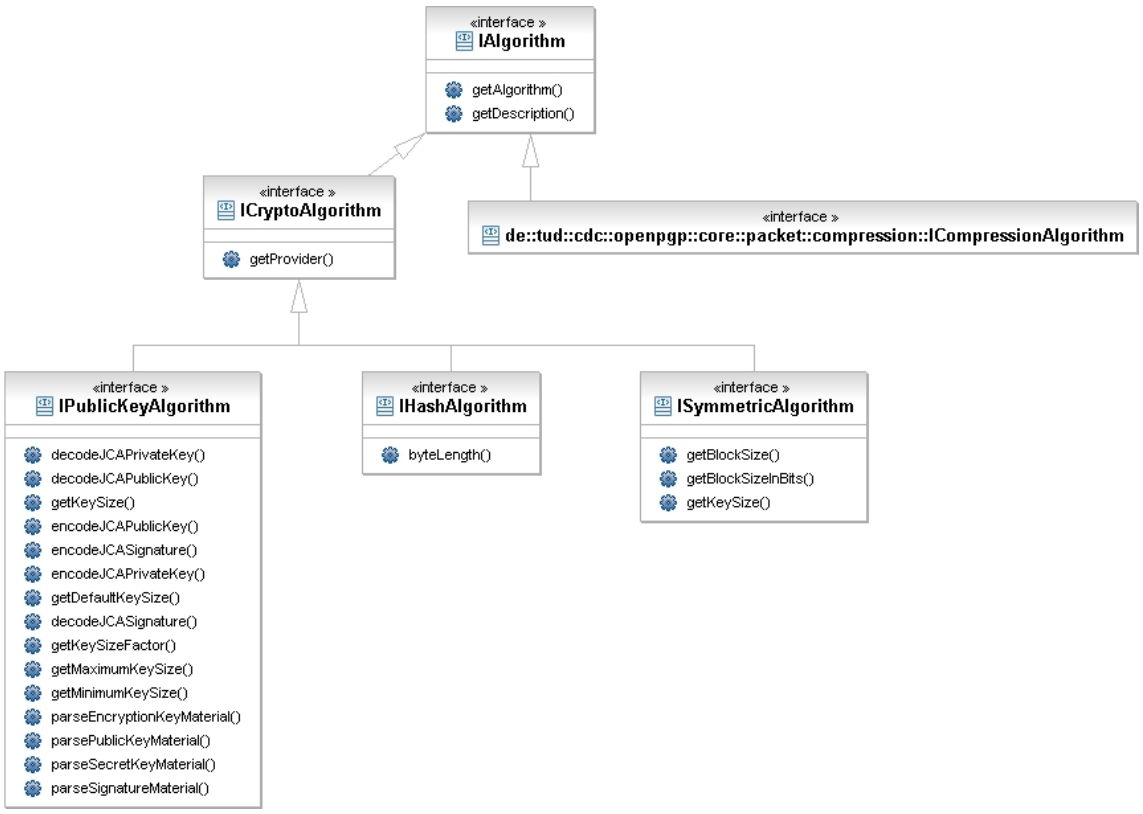

Abbildung 6.2: Vererbungshierarchie von `IAlgorithm`

Die Abbildung von OpenPGP Algorithmen IDs auf Algorithmenobjekte wird zentral in der Klasse `de.tud.cdc.openpgp.core.algorithm.CryptoAlgorithmFactory` durchgeführt[99]. Sie erfüllt den Zweck einer *Registrierungsklasse* (Registry). Es ist zu beachten, dass die symmetrischen Verfahren CAST5 und IDEA in der aktuellen Version der Bibliothek nicht unterstützt werden[100]. Der Zugriff auf ein Algorithmusobjekt mittels einer ID erfolgt dabei über die Methoden:

- `public static IPublicKeyAlgorithm getPublicKeyAlgorithm(int algorithm)`
- `public static ISymmetricAlgorithm getSymmetricAlgorithm(int algorithm)`
- `public static IHashAlgorithm getHashAlgorithm(int algorithm)`

[99] Kompressionsalgorithmen werden separat von der Klasse `CompressionAlgorithmFactory` im Paket `de.tud.cdc.openpgp.core.packet.compression` verwaltet.

[100] Die JCA implementiert weder CAST5 noch IDEA. Der FlexiProvider stellt zwar IDEA zur Verfügung, aber aufgrund einer Beschränkung bei der Verwendung des Algorithmus im CFB-Modus kann dieser nicht genutzt werden. Siehe hierzu auch den Quelltext der Klassen `de.tud.cdc.openpgp.core.algorithm.CryptoAlgorithmFactory` und `de.tud.cdc.openpgp.core.message.EncryptedMessage`.

Um unnötigen Speicherverbrauch zu vermeiden, werden die Algorithmenobjekte innerhalb von `CryptoAlgorithmFactory` als statische Variablen erzeugt. Zur Laufzeit existiert demnach jeweils nur ein Objekt pro Algorithmentyp.

Ein Nachteil der aktuellen Implementation von `CryptoAlgorithmFactory` ist die Tatsache, dass alle Algorithmendefinitionen explizit in der Klasse stehen. In einer späteren Version könnten diese beispielsweise in Property-Dateien ausgelagert und dann zur Laufzeit dynamisch erzeugt werden.

`CryptoAlgorithmFactory` verwaltet zudem die in der Anwendung zur Verfügung stehenden Schlüsselpaar-Typen. Ein Schlüsselpaar-Typ wird von der Klasse `KeyPairType` repräsentiert und ist meist eine Kombination aus einem asymmetrischen Hauptschlüssel-Paar (z.B. ein DSA Schlüssel) und einem zugehörigen Unterschlüsselpaar (z.B. ein ElGamal Schlüssel). Diese Objekte werden bei der Erzeugung eines neuen Schlüssels benötigt, um dem Anwender in der Benutzeroberfläche eine Liste der vorgegebenen Alternativen zu präsentieren. Der Zugriff auf die vordefinierten Schlüsselpaare erfolgt über die Methode `public static List<KeyPairType> getKeyPairTypes()`.

6.5.2 Nachrichtenverarbeitung

In Abschnitt 6.5 wurden bereits die verschiedenen Alternativen für ein mögliches Design der Nachrichtenverarbeitung diskutiert. Aus den genannten Gründen enthält die neue Bibliothek eine Lösung, die auf Datenströmen (Data Streams) basiert und somit primär auf den Klassen des `java.io` Pakets aufbaut.

6.5.2.1 Eingabe-/Ausgabeströme

Die abstrakte Superklasse aller byteorientieren Dateneingabeströme, ist die Klasse `InputStream`[101]. Sie definiert unter anderem Methoden, um Bytes aus dem Eingabestrom zu lesen (read) oder z.B. zu überspringen (skip). Ein Beispiel für eine Subklasse ist die Klasse `FileInputStream`, die Daten aus einer Datei liest. Das entsprechende Pendant zu `InputStream` für ausgehende Datenströme ist die Klasse `OutputStream`. Sie enthält Methoden, um Daten in den Ausgabestrom zu schreiben (write). Die Klasse FileOutputstream beispielweise schreibt Daten in eine Datei. Für zeichenorientiere (character based) Datenströme können Subklassen von `InputReader` und `OutputReader` verwendet werden.

Für die Entwicklung der Nachrichtenverarbeitung sind insbesondere die Klassen `FilterInputStream`[102], `FilterOutputStream`[103] und davon abgeleitete Klassen wichtig. Ein `FilterInput-/OutputStream` kapselt einen anderen `Input-/OutputStream` und fügt ihm neue Funktionalität hinzu. Dabei handelt es sich um das *Decorator*-Pattern (siehe

[101]Siehe http://java.sun.com/javase/6/docs/api/java/io/InputStream.html
[102]Siehe http://java.sun.com/javase/6/docs/api/java/io/FilterInputStream.html
[103]Siehe http://java.sun.com/javase/6/docs/api/java/io/FilterOutputStream.html

[FFS04]). Beispielsweise stellt das JDK die Klassen `DigestInputStream` und `CipherInputStream` für die Berechnung eines Hashwerts bzw. die Verschlüsselung eingehender Daten zur Verfügung. Eine OpenPGP Nachricht wird nun durch "Hintereinanderschalten" verschiedener Filter verarbeitet. Dies ist in Abbildung 6.3 illustriert. Erwähnenswert ist dabei insbesondere, dass die Nachrichtenverarbeitung in einem Durchgang (One-Time Pass) erfolgt, dass heisst sowohl beim Einlesen von Nachrichten, als auch bei der Erzeugung, werden keine temporären Dateien benötigt.

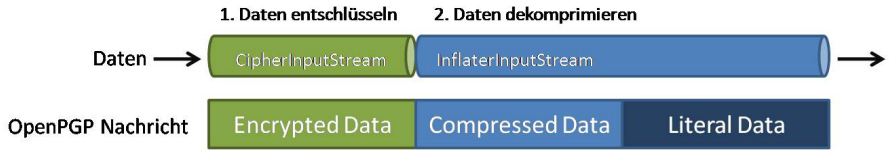

Abbildung 6.3: Beispiel für die Verarbeitung einer OpenPGP Nachricht mit Filtern

Die Klassen für die Eingabe-/Ausgabeverarbeitung liegen im Paket `de.tud.cdc.openpgp.core.io` und werden nun kurz beschrieben:

- `CipherInputStream`

  `CipherInputStream` wird für die Entschlüsselung von Daten verwendet. Der Stream wird mit einem `Cipher`[104]-Objekt initialisiert. Die Notwendigkeit diese Klasse zu schreiben, ergab sich aus einem Problem mit der Java `CipherInputStream`[105]-Implementation. Die darin enthaltene `read(byte[] b, int off, int len)`-Methode blockiert ggf. bis neue Daten vorhanden sind und liefert eventuell nicht alle angeforderten Daten in dem Array zurück.

- `DecompressionRepeater`

  Der `DecompressionRepeater` kapselt einen `InflaterInputStream`[106], also eine Klasse zum Dekomprimieren von Daten. Sie wird benötigt, da der `InflaterInputStream`, ähnlich wie der eben erwähnte `CipherInputStream`, nach dem Dekomprimieren von Daten blockiert und nicht (in einem Aufruf) die gewünschte Anzahl an Bytes zurückgibt. Die Klasse ruft daher solange die `read`-Methode des zugrundeliegenden Streams auf, bis alle angeforderten Daten zurückgeliefert wurden.

- `ExtendedDigestInputStream`

  Diese Klasse kapselt einen `DigestInputStream`[107] und erweitert ihn um die `skip`-Methode, zum Überspringen von Bytes innerhalb des Eingabestroms.

[104]Siehe http://java.sun.com/javase/6/docs/api/javax/crypto/Cipher.html
[105]Siehe http://java.sun.com/javase/6/docs/api/javax/crypto/CipherInputStream.html
[106]Siehe http://java.sun.com/javase/6/docs/api/java/util/zip/InflaterInputStream.html
[107]Siehe http://java.sun.com/javase/6/docs/api/java/security/DigestInputStream.html

- `Limiter`

  Intern verwenden die meisten Filterklassen einen Puffer (meist ein Array), um Daten für die Verarbeitung zwischenzuspeichern. OpenPGP Pakete haben allerdings (bis auf Ausnahmen) eine festgelegte Länge, die im Paketkopf (siehe Kapitel 4.1) angegeben wird. Ein Problem tritt dann auf, wenn ein Filter Daten aus einem "falschen" Pakettyp speichert. Wenn beispielsweise verschlüsselte Daten eines EncryptedData-Pakets, im Puffer eines Dekompressionsfilters zwischengespeichert werden, führt dies bei der weiteren Verarbeitung der Nachricht zu Fehlern. Der Einsatz der `Limiter`-Klasse soll dies verhindern. Der `Limiter` beschränkt die maximal erlaubte Anzahl von zu lesenden Bytes aus dem Eingabestrom. Im Normalfall wird er mit der Paketlänge initialisiert.

- `PacketHeaderLookAheadStream`

  Der `PacketHeaderLookAheadStream` sucht im Eingabestrom nach dem Auftreten eines bestimmten OpenPGP Pakets. Die Klasse wird bei Nachrichten, die Pakete unbestimmter Länge enthalten benötigt.

- `PartialHeaderInputStream`

  Implementiert das Einlesen eines partiell kodierten (siehe Kapitel 4.1) Datenpakets.

- `PartialHeaderOutputStream`

  Implementiert das Schreiben eines partiell kodierten Datenpakets.

- `Radix64InputStream`

  Implementiert einen `InputStream`, der Radix64-kodierte Daten aus einem zeichenorientierten Eingabestrom liest und dekodiert.

- `Radix64OutputStream`

  Diese Klasse kodiert Daten im Radix-64-Format.

- `SignatureInputStream`

  `SignatureInputStream` verifiziert eingehende Daten mit Hilfe eines `Signature`[108]-Objekts.

- `SignatureOutputStream`

  `SignatureOutputStream` signiert ausgehende Daten mit Hilfe eines `Signature`-Objekts.

Abbildung 6.4 zeigt die Hierarchie der, von `FilterInputStream` abgeleiteten Klassen.

[108]Siehe http://java.sun.com/javase/6/docs/api/java/security/Signature.html

Abbildung 6.4: Hierarchie der von `FilterInputStream` abgeleiteten Klassen

6.5.2.2 Nachrichten

Die in Kapitel 4.1.4 vorgestellten Nachrichtentypen werden von den Klassen im Paket `de.tud.cdc.openpgp.core.message` repräsentiert. Jede Nachricht implementiert das Interface `IMessage`. Es definiert die Methoden:

- `parse(PushbackInputStream in, MessageParser parser)`
- `write(OutputStream out, MessageCreator generator)`
- `getType()`

Das heisst ein bestimmtes Nachrichten-Objekt weiß, wie die zugehörige OpenPGP Nachricht eingelesen bzw. erzeugt wird. Es bedient sich dabei der verschiedenen OpenPGP Pakete (siehe Abschnitt 6.5.2.5). Die Klassen `MessageParser` (siehe Abschnitt 6.5.2.3) und `MessageCreator` (siehe Abschnitt 6.5.2.4) steuern dabei den Prozess der Nachrichtenverarbeitung und -erzeugung. Die Methode `getType()` dient der internen Identifikation des Nachrichtentyps. Die in der Bibliothek verwendeten Nachrichtentypen sind in Abbildung 6.5 aufgeführt. Hierbei ist zu beachten, dass auch transferierbare Schlüssel das `IMessage`-Interface implementieren und somit wie alle anderen Nachrichten verarbeitet werden.

Ein Problem ergab sich bei der Entwicklung der Klartext-Signatur (siehe Kapitel 4.3), die von der Klasse `ClearSignedMessage` repräsentiert wird. Eine Klartext-Signatur definiert die in den Signaturen verwendeten Hashalgorithmen vor dem Radix64-kodierten Text. Die Signaturen selbst, mit den Informationen über den für die jeweilige Signatur verwendeten Schlüssel, folgen nach dem Text. Die dahinterstehende Idee ist, dass die Hashwerte bei der Dekodierung des Textes direkt berechnet und für die Verifikation der

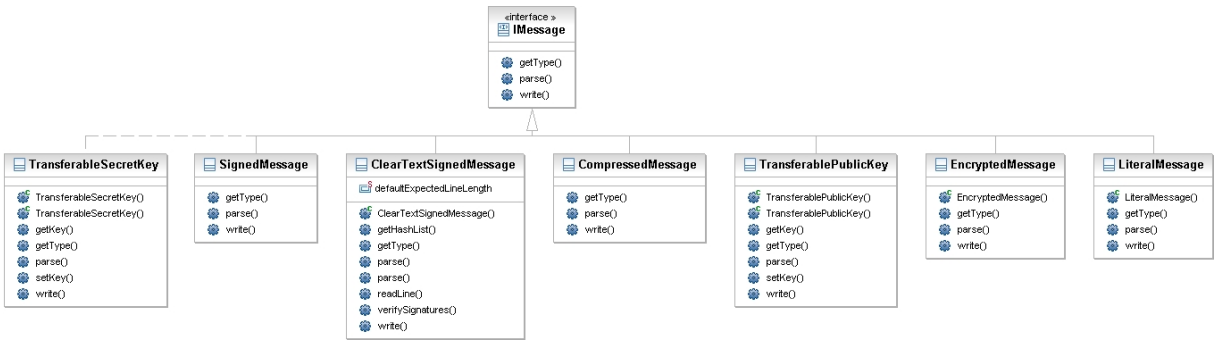

Abbildung 6.5: Vererbungshierarchie von `IMessage`

Signaturen benutzt werden. Leider können die für Signaturen verwendeten Objekte vom Typ `java.security.Signature` nicht mit einem bereits berechneten Hashwert initialisiert werden, sondern erwarten, je nach Betriebsmodus, einen privaten oder öffentlichen Schlüssel. Die Schlüsselkennungen und somit die Schlüsselobjekte stehen allerdings erst *nach* Bearbeiten des Radix64-kodierten Textes zur Verfügung. Dies hätte zur Folge, dass Nachrichten diesen Typs nicht in einem Durchgang (siehe Abschnitt 6.5.2.1) verarbeitet werden können. Um dies zu umgehen, werden Klartext-Signaturen nun zuerst rückwärts eingelesen, bis die notwendigen Signaturinformationen gefunden wurden. Im nächsten Schritt wird die Nachricht dann ganz normal von vorne nach hinten verarbeitet.

6.5.2.3 Einlesen von Nachrichten

Nachrichten werden über die Klasse `MessageParser` eingelesen. Sie definiert die Methoden `processEncryptedMessage`, `processCompressedMessage`, `processSignedMessage` und `processKeys`, die für die Verarbeitung von verschlüsselten, komprimierten und signierten Nachrichten, sowie von transferierbaren Schlüsseln verantwortlich sind. Intern werden beim Aufruf der Methoden die drei boolschen Statusvariablen `keysOnly`, `decrypt` und `verify` aktualisiert, die die erlaubten Nachrichtentypen beschränken. Ist beispielsweise die Variable `keysOnly` gesetzt, verarbeitet der `MessageParser` ausschließlich transferierbare Schlüssel. Danach rufen alle Methoden die Funktion `processMessages`() auf, die auf einer Menge von Quelldateien arbeitet. Pro Quelldatei wird der folgende rekursive Algorithmus durchlaufen:

1. Prüfe, ob die Datei mit einem gültigen Pakettyp beginnt.
2. Prüfe, ob die Datei binär oder Radix-64 kodiert ist und initialisiere je nach Ergebnis die IO-Objekte.
3. Prüfe, ob das Dateiende erreicht wurde.
4. Wenn nein, dann bestimme den Nachrichtentyp und erzeuge ein Nachrichtenobjekt des entsprechenden Typs.

5. Prüfe, ob der Nachrichtentyp für die aktuellen Einstellungen der Statusvariablen gültig ist.

6. Lese die Nachricht über das Nachrichtenobjekt ein.

7. Rekursiver Aufruf von Schritt 3.

Nachrichten werden mit der Klasse `MessageFactory` erzeugt, die das `Factory-Method`-Pattern [FFS04] nutzt.

6.5.2.4 Nachrichtenerstellung

Die Erzeugung von Nachrichten erfolgt mittels der Klasse `MessageCreator`, die analog zu der im vorherigen Abschnitt beschriebenen Klasse `MessageParser` funktioniert. Der `MessageCreator` wird über die boolschen Variablen `sign`, `compress`, `encrypt`, `asciiArmored` und `combineLiteralMessages` gesteuert. Die ersten drei Variablen geben an, ob Daten signiert, komprimiert und verschlüsselt werden sollen. Die `asciiArmored`-Variable bestimmt, ob die Daten Radix64-kodiert werden und `combineLiteralMessages` gibt an, ob die verarbeiteten Dateien in einer Zieldatei zusammengefasst werden. Die Klasse definiert die Methoden `createSymmetricallyEncryptedMessages`, `createSignedMessages`, `createCompressedMessages` und `createPublicKeyEncryptedMessages`, die für die Erzeugung der entsprechenden OpenPGP Nachrichten verantwortlich sind. Intern rufen die Methoden die Funktion `createMessages` auf, die auf einer Menge von Quelldateien arbeitet und den folgenden rekursiven Algorithmus für die Nachrichtenerstellung verwendet:

1. Prüfe, ob alle bearbeiteten Dateien in eine Zieldatei oder in mehrere Dateien geschrieben werden.

2. Erzeuge ggf. die Zielordnerstruktur.

3. Erzeuge anhand der aktuellen Einstellungen der Statusvariablen und des zuletzt verarbeiteten Nachrichtentyps die nächste Nachricht.

4. Schreibe die Nachricht.

5. Rekursiver Aufruf von Schritt 3.

6.5.2.5 OpenPGP Pakete

Die für die Verarbeitung der OpenPGP Pakete relevanten Klassen liegen im Paket `de.tud.cdc.openpgp.core.packet`. Folgende Einschränkungen liegen in der aktuellen Version der Bibliothek vor. Das Signieren von Schlüsseln der Version 3 ist zur Zeit nicht möglich. Die Interpretation der Signaturunterpakete wurde noch nicht vollständig implementiert. So werden momentan beispielsweise die Präferenzen für Hash- , Kompressions-

und symmetrischen Verschlüsselungsalgorithmus[109], sowie das Features-Unterpaket[110] ignoriert.

6.5.3 Schlüsselerzeugung

Schlüsselpaare können mit der Klasse `OPGPKeyPairGenerator` aus dem Paket `de.tud.cdc.openpgp.core.key` generiert werden. Hierfür steht die statische Methode `void generateKeyPair(KeyPairTypeGenerationParameters parameters, IProgressMonitor monitor)` zur Verfügung.

Das Parameterobjekt enthält folgende Werte:

- Typ des Schlüsselpaars

  Ein Objekt vom Typ `KeyPairType` (siehe Abschnitt 6.5.1).

- Schlüsselgröße

  Die Schlüsselgröße in Bit für den Haupt- und ggf. für den Unterschlüssel.

- Name und E-Mail Adresse

  Diese Angaben werden für die Erstellung des primären Schlüsselpaars verwendet.

- Ablaufdatum

  Das Ablaufdatum des generierten Schlüssels.

- Passphrase

  Die Passphrase, um den privaten Schlüssel zu verschlüsseln.

Der `IProgressMonitor`[111] wird für die Fortschrittsanzeige in der Oberfläche verwendet (siehe Abschnitt 6.6.3.1).

6.5.4 Schlüsselspeicherung

Für die Umsetzung der Persistenzschicht gab es zwei mögliche Ansätze. Die Schlüssel entweder in binären Dateien (diesen Ansatz verfolgt GnuPG) oder in einer relationalen Datenbank zu speichern. Da der OpenPGP Client in einer späteren Version eventuell um Suchfunktionen erweitert wird und die Datenbank Transaktionssicherheit bietet, wurde als Lösung die zweite Variante implementiert. Als Datenbank kommt *Apache Derby*[112] zum Einsatz, die mit Java Version 6 als *Java DB*[113] in das JDK integriert wurde. Der OpenPGP Client verwendet Derby[114] Version 10.3.1. Der Zugriff auf die Datenbank

[109] Siehe Siehe [CDF+07] Sektion 5.2.3.7., 5.2.3.8. und 5.2.3.9.
[110] Siehe [CDF+07] Sektion 5.2.3.24.
[111] Siehe http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api-/org/eclipse/core/runtime/IProgressMonitor.html
[112] http://db.apache.org/derby/
[113] http://developers.sun.com/javadb/
[114] Die Original Derby-Distribution ist meist aktueller als die Java DB Version, daher wurde dieser der Vorzug gegeben.

erfolgt über die Klassen aus dem `java.sql`-Paket.

Das Design der Datenbank-Klassen richtet sich nach dem J2EE *Data Access Object*-Pattern, welches ausführlich in [SM02] beschrieben wird. Die Hierarchie der Klassen ist in Abbildung 6.6 dargestellt. Alle relevanten Klassen und Interfaces liegen im Paket `de.tud.cdc.openpgp.core.db`.

Abbildung 6.6: Hierarchie der Datenbank-Klassen

Das Datenbankschema wird in der Datei `DB_SCHEMA.txt` beschrieben. Die einzelnen Tabellen werden im Folgenden erläutert:

PUBLIC_KEYS
Die Tabelle für die Speicherung der öffentlichen Haupt- und Unterschlüssel. Die öffentlichen Schlüssel selbst werden als BLOB[115] abgelegt. Die maximale Größe eines Schlüssels ist momentan auf 8KB beschränkt. Ein Subschlüssel referenziert einen Hauptschlüssel über das Attribut `PARENT_ID`. Um die Suche nach verschiedenen Schlüsselattributen zu ermöglichen, wurden diese in separate Spalten ausgelagert.

```
CREATE TABLE PUBLIC_KEYS (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
```

[115]BLOB steht für Binary Large Object, ein Objekt in dem große binäre Daten gespeichert werden.

```
  CONSTRAINT PUBLIC_KEYS_PK PRIMARY KEY,
  KEY_ID VARCHAR(16) NOT NULL,
  FINGERPRINT VARCHAR(40) NOT NULL,
  VERSION INT NOT NULL,
  ALGORITHM INT NOT NULL,
  KEY_SIZE INT NOT NULL,
  CREATION_DATE TIMESTAMP NOT NULL,
  EXPIRATION_DATE TIMESTAMP,
  PARENT_ID INT,
  PUBLIC_KEY BLOB(8K) NOT NULL
);
```

SECRET_KEYS
Die Tabelle speichert die verschlüsselten Daten von privaten Haupt- oder Unterschlüsseln. Jeder private Schlüssel muss einen öffentlichen Schlüssel aus der PUBLIC_KEYS-Tabelle referenzieren. Analog zum obigen Fall werden die privaten Schlüssel in BLOBs der maximalen Größe von 8KB abgelegt.

```
CREATE TABLE SECRET_KEYS (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT SECRET_KEYS_PK PRIMARY KEY,
  PUBLIC_KEY_ID INT NOT NULL,
  SECRET_KEY_DATA BLOB(8K) NOT NULL
);
```

REVOCATION_SIGNATURES
Diese Tabelle speichert die zu einem öffentlichen Schlüssel gehörenden Widerruf-Signaturen. Der zugehörige Schlüssel wird über eine ID referenziert. Die maximale Größe einer Widerruf-Signatur ist 8KB.

```
CREATE TABLE REVOCATION_SIGNATURES (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT REVOCATION_SIGNATURES_PK PRIMARY KEY,
  PUBLIC_KEY_ID INT NOT NULL,
  REVOCATION_SIGNATURE BLOB(8K) NOT NULL
);
```

DIRECT_SIGNATURES
Die Tabelle speichert die zu einem öffentlichen Schlüssel gehörenden direkten Signaturen. Der zugehörige Schlüssel wird über eine ID referenziert. Wie bei den Widerruf-Signaturen wird die Größe auf 8KB limitiert.

```
CREATE TABLE DIRECT_SIGNATURES (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT DIRECT_SIGNATURES_PK PRIMARY KEY,
  PUBLIC_KEY_ID INT NOT NULL,
```

```
  DIRECT_SIGNATURE BLOB(8K) NOT NULL
);
```

SUB_KEY_BINDING_SIGNATURES
Speichert die zu einem untergeordneten öffentlichen Schlüssel gehörenden Subkey-Binding Signaturen. Der zugehörige Schlüssel wird über eine ID referenziert. Die Größe der Signaturen wird auf 8KB limitiert.

```
CREATE TABLE SUB_KEY_BINDING_SIGNATURES (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT SUB_KEY_BINDING_SIGNATURES_PK PRIMARY KEY,
  PUBLIC_KEY_ID INT NOT NULL,
  SUB_KEY_BINDING_SIGNATURE BLOB(8K) NOT NULL
);
```

USER_IDS
Die Tabelle speichert die zu einem öffentlichen Schlüssel gehörenden Benutzerkennungen. Die maximale Länge einer Benutzerkennung beträgt momentan 2048 Zeichen. Der zugehörige Schlüssel wird über eine ID referenziert.

```
CREATE TABLE USER_IDS (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT USER_IDS_PK PRIMARY KEY,
  PUBLIC_KEY_ID INT NOT NULL,
  USER_ID VARCHAR(2048) NOT NULL
);
```

USER_ATTRIBUTES
Die Tabelle speichert die zu einem öffentlichen Schlüssel gehörenden UserAttributes. In der aktuellen Version handelt es sich dabei ausschließlich um Bilder, die eine Größe von 512KB nicht überschreiten dürfen. Der zugehörige Schlüssel wird über eine ID referenziert.

```
CREATE TABLE USER_ATTRIBUTES (
  ID INT NOT NULL GENERATED ALWAYS AS IDENTITY
  CONSTRAINT USER_ATTRIBUTES_PK PRIMARY KEY,
  PUBLIC_KEY_ID INT NOT NULL,
  USER_ATTRIBUTE BLOB(512) NOT NULL
  );
```

Erzeugen der Foreign Keys
Erzeugt die in den Tabellen verwendeten Spaltenreferenzen.

```
ALTER TABLE PUBLIC_KEYS ADD CONSTRAINT
PARENT_ID_FK Foreign Key (PARENT_ID) REFERENCES PUBLIC_KEYS (ID)
ON DELETE CASCADE ON UPDATE RESTRICT;
```

ALTER TABLE SECRET_KEYS ADD CONSTRAINT SECRET_KEY_PUBLIC_KEY_FK
Foreign Key (PUBLIC_KEY_ID) REFERENCES PUBLIC_KEYS (ID)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE REVOCATION_SIGNATURES ADD CONSTRAINT
REVOCATION_SIGNATURE_PUBLIC_KEY_FK Foreign Key (PUBLIC_KEY_ID)
REFERENCES PUBLIC_KEYS (ID) ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE DIRECT_SIGNATURES ADD CONSTRAINT
DIRECT_SIGNATURE_PUBLIC_KEY_FK Foreign Key (PUBLIC_KEY_ID)
REFERENCES PUBLIC_KEYS (ID) ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE SUB_KEY_BINDING_SIGNATURES ADD CONSTRAINT
SUB_KEY_BINDING_SIGNATURE_PUBLIC_KEY_FK Foreign Key (PUBLIC_KEY_ID)
REFERENCES PUBLIC_KEYS (ID) ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE USER_IDS ADD CONSTRAINT USER_ID_PUBLIC_KEY_FK
Foreign Key (PUBLIC_KEY_ID) REFERENCES PUBLIC_KEYS (ID)
ON DELETE CASCADE ON UPDATE RESTRICT;

ALTER TABLE USER_ATTRIBUTES ADD CONSTRAINT
USER_ATTRIBUTES_PUBLIC_KEY_FK Foreign Key (PUBLIC_KEY_ID)
REFERENCES PUBLIC_KEYS (ID) ON DELETE CASCADE ON UPDATE RESTRICT;

Erzeugen der Indizes
Eine Reihe von Indizes, die die Suche nach öffentlichen Schlüsseln beschleunigen.

CREATE INDEX KEYS_ID_IDX ON PUBLIC_KEYS (ID);
CREATE INDEX KEY_ID_IDX ON PUBLIC_KEYS (KEY_ID);
CREATE INDEX FINGERPRINT_IDX ON PUBLIC_KEYS (FINGERPRINT);
CREATE INDEX USER_ID_IDX ON USER_IDS (USER_ID);

6.5.5 Schlüsselimport/-export

Schlüssel können entweder aus einer Datei oder von einem Schlüsselserver importiert, bzw. in eine Datei oder auf einen Schlüsselserver exportiert werden. Die verschiedenen Möglichkeiten werden im Folgenden beschrieben.

Schlüsselimport aus einer Datei
Da ein transferierbarer Schlüssel das `IMessage`-Interface implementiert, kann er einfach mit Hilfe des `MessageParsers` (siehe Abschnitt 6.5.2.3) eingelesen werden. Hierfür steht die Methode `processKeys(String sourceFile)` zur Verfügung, die als Parameter den Namen der Datei erwartet, in der sich die Schlüssel befinden. Sie unterstützt die Verarbeitung von privaten und öffentlichen Schlüsseln. Die eingelesenen Schlüssel werden für die Weiterverarbeitung im `MessageParser` gespeichert.

Schlüsselexport in eine Datei
Für den Export eines Schlüssels wird dementsprechend die Klasse `MessageCreator` (siehe Abschnitt 6.5.2.4) verwendet, die die Methode `writeKeys(String destinationFile, List<ITransferableKey>, boolean asciiArmored)` definiert.

Schlüsselimport von einem Server
Die grundlegenden Operationen auf einem Schlüsselserver sind die Suche, der Import und der Export von Schlüsseln. Diese Basisfunktionen definieren die Schnittstelle eines Schlüsselserver-Protokolls, die in dem Interface `IKeyServerProtocol` beschrieben wird.

Es enthält die folgenden Methoden:

- `List<PublicKeyInformation> searchKeys(String keyWords)`

  Führt eine Suche auf dem Schlüsselserver mit den angegebenen Suchwörtern aus und gibt eine Liste mit Informationen über die gefundenen öffentlichen Schlüssel zurück. Das `PublicKeyInformation`-Objekt enthält unter anderem die Schlüsselkennung, den Typ des Schlüssels, die Schlüsselgröße, sowie eine Liste von Benutzerkennungen.

- `List<ITransferablePublicKey> importKeys(List<PublicKeyInformation> keys)`

  Importiert öffentliche Schlüssel von einem Schlüsselserver. Als Parameter enthält die Methode eine Liste von `PublicKeyInformation`-Objekten, die als Ergebnis einer Suche zurückgegeben wurden. Die importierten Schlüssel werden als Liste vom Typ `ITransferablePublicKey` (siehe Abschnitt 6.5.2.2) zurückgegeben.

- `void exportKeys(List<ITransferablePublicKey> keys)`

  Exportiert die übergebene Liste von öffentlichen Schlüsseln zu einem Schlüsselserver.

- `boolean maximumHitsExceeded()`

  Prüft, ob bei einer Suche die maximale Trefferanzahl erreicht wurde.

Das Interface liegt im Paket `de.tud.cdc.openpgp.core.protocol`. Momentan unterstützt die Bibliothek den Zugriff auf einen Schlüsselserver über das OpenPGP *HTTP Keyserver Protocol* (HKP) [Sha03]. Das Protokoll wird in der Klasse `HttpKeyServerProtocol` umgesetzt, die `IKeyServerProtocol` implementiert. Die maximale Trefferliste ist beim HKP momentan auf 100 Treffer beschränkt.

Die Verwendung eines abstrakten Interfaces als Beschreibung für den Zugriff auf einen Schlüsselserver eignet sich für die spätere Einführung von neuen Protokollen, die dann ggf. zur Laufzeit ausgewählt werden können (Strategy-Pattern).

Schlüsselexport zu einem Server
Der Export zu einem Schlüsselserver ist in der aktuellen Version nicht implementiert.

6.5.6 Vertrauensmodelle

Der OpenPGP Client enthält eine Implementierung des in Kapitel 3.2 vorgestellten Web of Trust. Diese findet sich in der Klasse `de.tud.cdc.openpgp.core.trust.WebOfTrustModel`. Als Standardparameter für das Web of Trust werden die Vorgaben von GnuPG verwendet, mit `COMPLETES_NEEDED`=1, `MARGINALS_NEEDED`=3 und einem maximalen Zertifizierungspfad der Länge 5. In der aktuellen Version der Bibliothek werden die Vertrauens- und Gültigkeitswerte noch nicht in der Datenbank gespeichert, somit erfolgt die Berechnung der Schlüsselvalidität nur auf Basis von temporären Vertrauenswerten. Diese Werte werden daher noch nicht konkret in der Anwendung genutzt.

Damit der OpenPGP Client möglichst einfach um alternative Vertrauensmodelle, wie z.B. das Probabilistische Modell (siehe Kapitel 3.3) erweitert werden kann und eine dynamische Zuweisung eines Vertrauensmodells zur Laufzeit möglich ist, wird, wie schon bei der Umsetzung der Algorithmen-Objekte, das Strategy-Pattern verwendet. Jede Klasse, die ein Vertrauensmodell repräsentiert, muss das Interface `de.tud.cdc.openpgp.core.trust.ITrustModel` und die darin definierte Methode `void computeKeyValidity(List<ITransferableKey> publicKeys)` implementieren, die eine Berechnung der Gültigkeitswerte für die übergebene Schlüsselliste auf Basis des jeweiligen Vertrauensmodells durchführt (siehe Abbildung 6.7[116]).

6.6 OpenPGP Client Benutzeroberfläche (UI)

Das Plug-in `de.tud.cdc.openpgp.ui` implementiert die Benutzeroberfläche des OpenPGP Clients und definiert mit der Klasse `Application` den Einstiegspunkt in die Anwendung, die beim Start von Eclipse aufgerufen wird. Zusätzlich enthält es in der Datei *openpgp.product* die Produktdefinition (siehe Kapitel 5.4) des OpenPGP Clients. Dort werden unter anderem die Brandinginformationen wie Splash-Screen, About-Dialog, etc. konfiguriert. Aufgrund der umfangreichen Änderungen an der OpenPGP Bibliothek werden in der Oberfläche momentan nur einige Basisfunktionen unterstützt. Die verschiedenen Teilbereiche des UI werden im Folgenden ausführlich dargestellt.

6.6.1 Aufbau des Hauptfensters

Das Hauptfenster der Anwendung setzt sich aus folgenden Komponenten zusammen:

Menü
Das Menü erlaubt den Zugriff auf die Benutzerpräferenzen, das Benutzerhandbuch des OpenPGP Clients, den About-Dialog, sowie das Schließen der Anwendung. Die Menüeinträge, sowie die beim Klicken auf einen Menüeintrag ausgeführten Actions (siehe Kapitel

[116] Das probabilistische Modell wird nicht unterstützt, wurde allerdings in der Rahmenklasse `ProbabilisticModel` für die spätere Entwicklung vorgesehen.

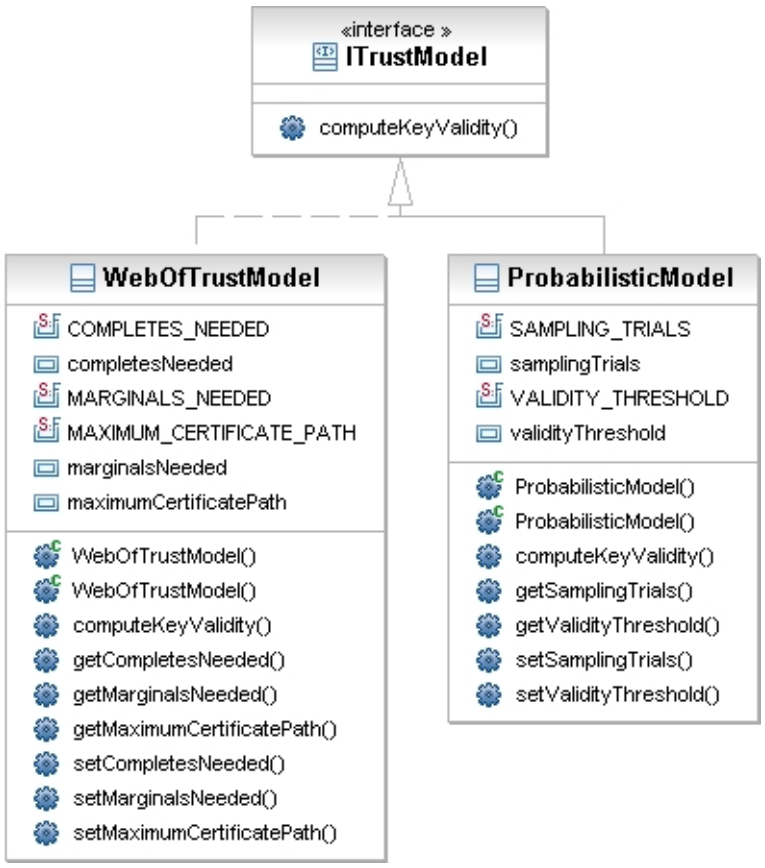

Abbildung 6.7: Vererbungshierarchie von `ITrustModel`

5.1), werden in den Methoden `makeActions(IWorkbenchWindow window)` und `fillMenuBar(IMenuManager menuBar)` der Klasse, `de.tud.cdc.openpgp.ApplicationActionBarAdvisor` erzeugt, die beim Start der Anwendung automatisch von der Workbench aufgerufen werden.

Werkzeugleiste
Die Werkzeugleiste (Toolbar) enthält die Hauptfunktionen der Anwendung. Sie erlaubt das Erzeugen neuer asymmetrischer Schlüsselpaare, die Verschlüsselung und Entschlüsselung, sowie das Signieren und Verifizieren von Daten und schließlich den Import/Export von Schlüsseln und die Auswahl der aktuell in der Benutzeroberfläche verwendeten Sprache. Die, der Werkzeugleiste zugrundeliegenden Actions werden, wie auch die im Menü verwendeten Actions, innerhalb der Methode `makeActions` im `ApplicationActionBarAdvisor` erzeugt. Das Aussehen der Werkzeugleiste wurde speziell für die Anwendung angepasst (siehe Abschnitt 6.6.2.1), daher werden die Toolbar-Einträge nicht innerhalb der Standardmethode `fillCoolBar(ICoolBarManager coolBar)` im `ApplicationActionBarAdvisor` hinzugefügt, sondern stattdessen in der Methode `fillCustomizedToolBar(ToolBarManager toolBarManager)`.

Schlüsselübersicht
Die Schlüsselübersicht wird im View `de.tud.cdc.openpgp.ui.views.KeysView` implementiert. Die grundlegende Idee für die Umsetzung dieses Views ist, dass der Anwender durch alle in der Datenbank gespeicherten öffentlichen Schlüssel mit den zugehörigen Bestandteilen wie Signaturen, Benutzeridentitäten, Fotos etc. navigieren und die Eigenschaften des aktuell selektierten Objekts betrachten kann.

Eclipse stellt für diesen Anwendungsfall den sogenannten `Master/Details-Block`[117] aus dem Paket `org.eclipse.ui.forms` zur Verfügung. Der `Master/Details-Block` unterteilt den View in einen linken Bereich, in dem die Masterobjekte zum Beispiel in Form einer Liste oder Baumstruktur dargestellt werden und einen rechten Detailbereich, der die Eigenschaften eines selektierten Masterobjekts anzeigt[118]. Pro Objekttyp, der in der Masterliste vorkommt, wird eine Detailseite innerhalb des `Master/Details-Block` registriert. Bei Selektion eines Masterobjekts, wird die Detailansicht mit der zugehörigen Seite aktualisiert. Der im View dargestellte `Master/Details-Block` wird von der Klasse `de.tud.cdc.openpgp.ui.views.KeysMasterDetails` implementiert. Der konkrete Masterbereich zeigt die Schlüsselliste in Form einer Baumstruktur an. Intern wird hierfür die Klasse `org.eclipse.jface.viewers.TreeViewer`[119] genutzt, die wiederum auf der SWT `Tree`-Komponente aufbaut. Im rechten Bereich werden die folgenden Detailseiten angezeigt, die im Paket `de.tud.cdc.openpgp.ui.views` liegen:

- KeyDetailsPage

  Stellt die Eigenschaften eines öffentlichen Hauptschlüssels dar.

- SubKeyDetailsPage

  Stellt die Eigenschaften eines Unterschlüssels dar.

- SignatureDetailsPage

  Zeigt die Eigenschaften einer Signatur an.

- UserIdDetailsPage

  Zeigt die Eigenschaften einer Benutzerkennung an.

Bei Erzeugung des Views wird der `Master/Details-Block` in ein `Eclipse Form` (siehe Abschnitt 6.6.2.3) eingebettet.

[117] Siehe hierzu [Glo05]

[118] Intern legt der `Master/Details-Block` die beiden Bereiche in einem `org.eclipse.swt.custom.SashForm` (siehe [ecl]) aus. Ein `SashForm` ist ein Containerobjekt, welches darin liegende Unterkomponenten in einem horizontalen oder vertikalen Layout darstellt. Die einzelnen Komponenten werden dabei jeweils von einem Trenner, dem sogenannten `Sash` unterteilt, der es erlaubt die Größe der Komponenten mit der Maus zu ändern.

[119] Siehe http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api-/org/eclipse/jface/viewers/TreeViewer.html.

Aktionen auf einzelnen Masterobjekten werden über ein Kontextmenü aufgerufen, das über die rechte Maustaste geöffnet werden kann. Ein öffentlicher Schlüssel könnten beispielsweise in die Zwischenablage kopiert oder gelöscht werden. Das Kontextmenü wird in der aktuellen Version bereits angezeigt, aus zeitlichen Gründen konnte jedoch die Funktionalität der Menüpunkte nicht mehr implementiert werden.

Ereignisprotokollanzeige
Dieser View wird für die Anzeige von aufgetretenen Ereignissen, wie Warnungen oder Fehlern verwendet. Es handelt sich dabei um den Standard *ErrorLog View* von Eclipse (siehe Abschnitt 6.9). Um den Fokus auf die Schlüsselübersicht zu lenken, ist die Ereignisprotokollanzeige beim Start des Programms minimiert, dass heißt es ist nur die Titelleiste des Views sichtbar.

Der OpenPGP Client verwendet nur eine Perspektive (siehe Kapitel 5.2), die das Layout des Fensters und die Positionierung der Views vorgibt. Sie ist in der Datei `de.tud.cdc.openpgp.ui.Perspective` definiert und wird in der, von der Workbench aufgerufenen `ApplicationWorkbenchAdvisor` Klasse, als initiale Perspektive gesetzt.

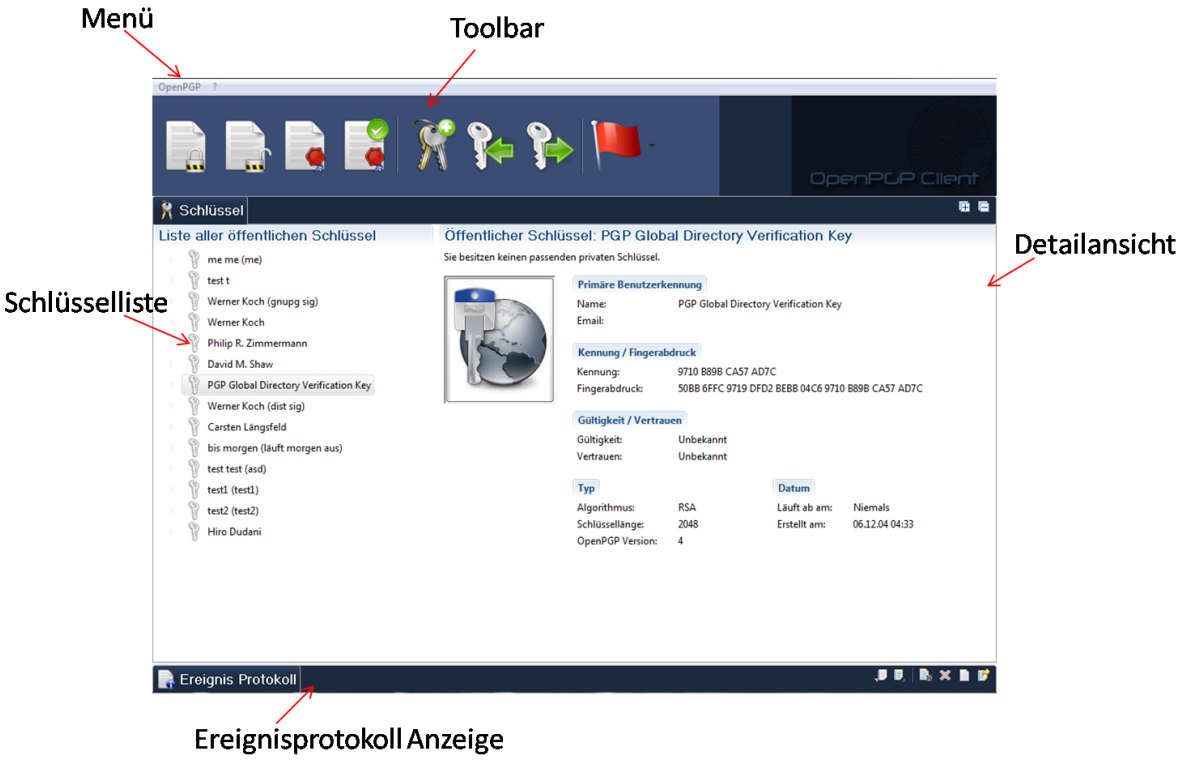

Abbildung 6.8: Aufbau des Hauptfensters

6.6.2 Individuelle Anpassung der Eclipse-Oberfläche

Ein Ziel beim Entwurf der Benutzeroberfläche des OpenPGP Client war die einfache Bedienbarkeit der Anwendung, um auch Anwendern ohne kryptographische Kenntnisse eine möglichst intuitive Nutzung der verschiedenen Funktionen zu erlauben. Im Standardfall erbt eine RCP-Anwendung das Look&Feel von Eclipse, das heisst die Oberfläche sieht genau so aus und verhält sich wie Eclipse selbst. Um dem OpenPGP Client ein individuelles Aussehen zu verleihen, welches sich von der typischen Eclipse Oberfläche abhebt, wurde die Werkzeugleiste (Toolbar), sowie die zugrundeliegende Oberflächenpräsentation durch ein neues Design (Farbe, Icons, etc.) ersetzt. Zudem bietet die Workbench im Standardfall zahlreiche Funktionen, die einen ungeübten Anwender eher überfordern. So können beispielsweise Views per Drag&Drop verschoben, geschlossen, minimiert, maximiert oder vom Workbenchfenster entkoppelt werden. Um die Arbeit mit dem OpenPGP Client zu erleichtern und Anwenderfehler zu minimieren, wurde dieses Standardverhalten daher durch Anpassung der Präsentation überschrieben. Die durchgeführten Änderungen und verwendeten Iconsets werden nun näher beschrieben.

6.6.2.1 Werkzeugleiste

Beim Starten von Eclipse erzeugt die Workbench das Hauptfenster der Anwendung. Dies geschieht in der Methode `public void createWindowContents(final Shell shell)` in der Klasse `ApplicationWorkbenchWindowAdvisor`. Im Standardfall generiert die Methode die Menü-, Werkzeug-, Perspektiv- und Schnellansichtsleiste (fast view bar), die Statuszeile sowie einen Bereich für eine Fortschrittsanzeige. Die Perspektivleiste erlaubt die Auswahl der aktiv verwendeten Perspektive. Die Schnellansichtsleiste stellt einen View in minimierter Form, repräsentiert durch ein Icon dar, so dass der View jederzeit über die Leiste geöffnet werden kann. Um nun die Werkzeugleiste durch ein eigenes Design zu ersetzen, wurde die `createWindowContents`-Methode überschrieben. Die daraus resultierende Fensterstruktur besteht aus der Menüleiste, einem Banner, dass die angepasste Werkzeugleiste, sowie mehrere Hintergrundgraphiken enthält und dem Seitenbereich, in dem Views, Editoren etc. angezeigt werden können (siehe hierzu auch Abschnitt 6.6.1). Die Werkzeugleiste erlaubt den Zugriff auf die Hauptfunktionen (siehe Abschnitte 6.6.1 und 6.6.3) des OpenPGP Clients. Um den Fokus des Anwenders auf diese Funktionen zu richten, werden sie durch extra große Icons repräsentiert. Die Werkzeugleiste ist in Abbildung 6.9 dargestellt.

Es ist zu beachten, dass aufgrund eines zur Zeit noch offenen Fehlers[120] in Eclipse, die Perspektiv- und Schnellansichtsleiste, sowie die Fortschrittsanzeige, beim Überschreiben der `createWindowContents`-Methode nicht erzeugt werden können. Für den Fall, dass diese Komponenten benötigt werden, muss auf das Standard-Fensterlayout zurückgegriffen werden.

[120] Siehe https://bugs.eclipse.org/bugs/show_bug.cgi?id=73821

Abbildung 6.9: Die Werkzeugleiste des OpenPGP Clients

6.6.2.2 Präsentation

In Kapitel 5.2 wurde das Konzept von Perspektiven erläutert. Eine Perspektive arrangiert Views und Editoren in Form von sogenannten Stapeln (Stacks), die Objekte gleicher Art zusammenfassen. So kann ein Stack beispielsweise mehrere Views enthalten, zwischen denen per Reiter (Tab) gewechselt werden kann. Es gibt drei verschiedene Stapeltypen: View, StandaloneView und Editor.

Das Aussehen und Verhalten dieser Stapel wird von einer **Präsentation** (presentation) bestimmt. Um das Look&Feel des OpenPGP Clients anzupassen, wurde eine eigene Präsentation implementiert. Sie befindet sich im Plug-in `de.tud.cdc.openpgp.presentation`. Zunächst muss dafür der Extension Point `org.eclipse.ui.presentationFactories` erweitert werden. Der Extension Point deklariert eine Factory-Klasse `OpenPGPClientPresentationFactory`, welche die Klasse `AbstractPresentationFactory` erweitert. Die Factory-Klasse wird von der Workbench für die Erzeugung der unterschiedlichen Stapeltypen aufgerufen und generiert pro Typ eine sogenannte `StackPresentation`. Da in der aktuellen Version des OpenPGP Clients ausschließlich Views zum Einsatz kommen, verwendet die Präsentation für alle drei Typen dieselbe Stackpresentation `OpenPGPClientPartPresentation`. Sollten sich die Anforderungen an die Software in Zukunft ändern, muss die Präsentation ggf. angepasst werden.

Die Präsentation setzt unter anderem folgende Änderungen um:

- Anpassen der Titelleiste von Views
- Anpassen der in der Anwendung verwendeten Farben
- Views können nicht mehr minimiert, maximiert, geschlossen oder per Drag&Drop verschoben werden

Die Präsentation wird in der Datei `plugin_customization.ini` im UI-Plug-in als Standard-Präsentation registriert.

Die in der Präsentation benutzten Farbwerte wie z.B die Hintergrundfarbe von Wizards, Farbe der View-Titelleiste, Hintergrundfarbe der Werkzeugleiste etc. werden in einem sogenannten **Thema** definiert, das den Extension Point `org.eclipse.ui.themes` erweitert. Das Thema wird mit allen Farbwerten in der `plugin.xml`-Datei des Präsentations-Plug-ins definiert und in der Datei `plugin_customization.ini` im UI-Plug-in als Standard-Thema für den OpenPGP Client registriert.

6.6.2.3 Eclipse Formulare

Formulare (Forms) wurden mit Version 3.0 von Eclipse eingeführt und bauen auf den Klassen von SWT und JFace auf. Die Grundidee für die Entwicklung von Eclipse Forms war, den vom Betriebssystem abhängigen nativen Look von Eclipse zu erweitern, um die Entwicklung von "[...] portablen web-orientierten Benutzeroberflächen [zu ermöglichen] [...]"[Glo05].

Zu den Erweiterungen gehören unter anderem [Glo05]:

- Eine Formularkomponente (Form), die in Views, Editoren etc. eingebettet werden kann.
- Verwaltung der im Form verwendeten Farben.
- Auslegen von Formularkomponenten vergleichbar mit einer Tabelle in HTML.
- Neue, in Formularen einsetzbare Komponenten, wie Sektionen, scrollbare Bereiche, Hyperlinks etc.

Alle im OpenPGP Client verwendeten Wizards (siehe Abschnitt 6.6.3), sowie der View für die Schlüsselübersicht (siehe Abschnitt 6.6.1) legen ihre Komponenten mit Hilfe von Formularen aus. Für eine ausführlichere Beschreibung von Formularen siehe [Ani07] und [Glo05].

6.6.2.4 Icons

Um den OpenPGP Client von dem normalen Eclipse Aussehen abzuheben, wurden Teile, der in der Anwendung verwendeten Icons ersetzt. Der OpenPGP Client verwendet Icons aus den folgenden Icon-Sets (die jeweilige Lizenz unter der die Icons veröffentlich wurden ist, sofern bekannt, in Klammern angegeben):

- Eclipse Standard Icons (EPL[121])
- Snowish 1.3[122] (GPL[123])
- NuoveXT 2.2[124] (LGPL[125])
- Silk[126] (Creative Commons Attribution 2.5[127])
- Isoglass (LGPL)

[121] Die Icons stehen vermutlich wie Eclipse selbst unter der Eclipse Public License (EPL). Dies war aber nicht konkret zu ermitteln. Siehe auch http://www.eclipse.org/org/documents/epl-v10.php
[122] http://www.kde-look.org/content/show.php?content=42905
[123] GNU General Public License. Siehe http://www.fsf.org/licensing/licenses/gpl.html
[124] http://www.kde-look.org/content/show.php/nuoveXT+2?content=62630
[125] GNU Lesser General Public License. Siehe http://www.fsf.org/licensing/licenses/lgpl.html
[126] http://www.famfamfam.com/lab/icons/silk/
[127] Siehe http://creativecommons.org/licenses/by/2.5/

- Crystal Project[128] (LGPL)
- Crystal Diamond[129] (GPL)
- Tango Iconset[130] (Creative Commons Attribution Share-Alike 2.5[131])

Die Icons wurden zum Teil mit Bildbearbeitungssoftware an die jeweilige Funktion in der Anwendung angepasst.

6.6.3 Darstellung der Hauptfunktionen

In diesem Abschnitt werden die, über die Toolbar verfügbaren Hauptfunktionen des OpenPGP Clients, näher beschrieben. Die meisten Funktionen verwenden dabei einen Assistenten (Wizard), der den Benutzer über mehrere Seiten durch einen bestimmten Prozess, z.B. die Erstellung eines neuen Schlüsselpaars, führt. JFace stellt hierfür die Basisklasse `org.eclipse.jface.wizard.Wizard`[132] zur Verfügung, die von den Assistenten für die jeweilige Funktion erweitert und angepasst wird. Die einzelnen Seiten werden beim Wizard registriert und erweitern die Klasse `org.eclipse.jface.wizard.WizardPage`[133]. Alle Wizards, sowie die Wizardseiten liegen im Paket `de.tud.cdc.openpgp.ui.wizards`.

6.6.3.1 Schlüsselerstellung

Ein neues Schlüsselpaar wird mit Hilfe des `CreateKeyPairWizard` erzeugt. Der Wizard besteht aus zwei Seiten, `CreateKeyPairPage1` und `CreateKeyPairPage1`. Auf der ersten Seite kann der Anwender folgende Einstellungen vornehmen:

- Name und E-Mail-Adresse für die Erzeugung der primären Benutzerkennung
- Schlüsselpaar-Typ (z.B. DSA Hauptschlüssel, RSA Subschlüssel)
- Schlüsselgröße in Bit
- Gültigkeitsdauer des Schlüsselpaars

Auf der zweiten Seite kann der Anwender die Passphrase festlegen, mit welcher der geheime Teil des Schlüsselpaars geschützt wird. Die Sicherheit des gewählten Passworts wird dem Anwender über eine Fortschrittsanzeige signalisiert. Der Wert, der dieser

[128] http://www.everaldo.com/crystal/
[129] http://www.kde-look.org/content/show.php/Crystal+Diamond+Icons?content=45576
[130] http://tango.freedesktop.org/Tango_Desktop_Project
[131] http://creativecommons.org/licenses/by-sa/2.5/
[132] http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api-/org/eclipse/jface/wizard/Wizard.html
[133] http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.isv/reference/api-/org/eclipse/jface/wizard/WizardPage.html

Fortschrittsanzeige zugrunde liegt, wird über eine modifizierte Version des Open Source Programms *PasswordCheck*[134] ermittelt, das eine Gewichtung der Passphrase anhand der eingegebenen Zeichentypen (z.B. Sonderzeichen, Klein- und Großschreibung, etc.) berechnet. Die auf den Seiten eingegeben Daten bzw. ausgewählten Werte, werden im Wizard in einem `KeyPairTypeGenerationParameters`-Objekt zwischengespeichert. Hat der Anwender alle notwendigen Eingaben durchgeführt, erzeugt der Wizard das Schlüsselpaar über die `generateKeyPair`-Methode der `OPGPKeyGenerator`-Klasse (siehe Abschnitt 6.5.3). Der Methode wird das Modelobjekt mit den gespeicherten Werten, sowie ein Progressmonitor für die Aktualisierung der Fortschrittsanzeige im Wizard mitgegeben. Nach erfolgreicher Erzeugung des Schlüsselpaars, wird es in der Datenbank gespeichert.

6.6.3.2 Daten verschlüsseln

Daten können entweder symmetrisch mit einer Passphrase oder asymmetrisch mit öffentlichen Schlüsseln verschlüsselt werden. In beiden Fällen öffnet sich der `EncryptDataWizard`, lediglich die angezeigten WizardPages variieren je nach Auswahl. Bei der symmetrischen Verschlüsselung wählt der Anwender auf der ersten Seiten den Zielordner, sowie die Dateien die verschlüsselt werden sollen. In den erweiteren Optionen kann der Anwender folgende Optionen anpassen (in Klammern ist der Standardwert angegeben):

- Verschlüsselungsverfahren (AES256)
- Kompression aktivieren/deaktivieren (aktiv)
- Kompressionsverfahren (ZIP)
- Signieren aktivieren/deaktivieren (aktiv)

Wurde das Signieren der Dateien aktiviert, muss auf der nächsten Seite der Signaturschlüssel ausgewählt und die zugehörige Passphrase zur Entsperrung des Schlüssels eingegeben werden. Nach Abschließen des Wizards werden die Dateien mit dem `MessageCreator` (siehe Abschnitt 6.5.2.4) verschlüsselt. Der Ablauf für die Verschlüsselung mit öffentlichen Schlüsseln ist fast identisch. Auf der ersten Seite müssen zusätzlich die öffentlichen Schlüssel ausgewählt werden, mit denen die Dateien verschlüsselt werden sollen. Weiterhin entfällt die Wahl des Verschlüsselungsverfahrens in den Optionen.

6.6.3.3 Daten entschlüsseln

Wenn der Anwender die Datenentschlüsselung ausgewählt hat, öffnet sich ein Dialog, in dem die zu entschlüsselnden OpenPGP Nachrichten im Dateisystem selektiert werden können. Nach einem Klick auf den Schalter *OK*, wird die Entschlüsselung über den `MessageParser` (siehe Abschnitt 6.5.2.3) durchgeführt. Der OpenPGP Client prüft selbstständig, ob die jeweilige Datei symmetrisch oder asymmetrisch verschlüsselt wurde. Wenn eine Passphrase erforderlich ist, erscheint ein Dialog, der den Benutzer zur

[134] http://justwild.us/examples/password/

Eingabe auffordert. Da die Dateien durchaus durch unterschiedliche Passphrasen bzw. öffentliche Schlüssel gesichert sein können, werden die eingegebenen Passwörter zwischengespeichert. Somit können Mehrfacheingaben vermieden und die Anzahl notwendiger Interaktionen seitens des Anwenders minimiert werden. Ein Statusfenster informiert den Anwender über den Fortschritt der Entschlüsselung.

6.6.3.4 Daten signieren

Das Signieren von Daten wird über den `SignDataWizard` abgewickelt. Er enthält nur die Seite `SignDataPage1`. Der Anwender muss zunächst den Zielordner festlegen, in dem die signierten Dateien abgelegt werden. Danach selektiert er die zu signierenden Dateien im Dateisystem. Im letzten Schritt wählt er aus den vorhandenen privaten Schlüsseln den zu verwendenden Signaturschlüssel aus und gibt die Passphrase ein, welche den privaten Schlüssel entsperrt. Optional kann ausgewählt werden, ob die Dateien zusätzlich komprimiert werden und welches Kompressionsverfahren dafür angewendet wird. Nach Abschließen des Wizards werden die signierten Dateien mit dem `MessageCreator` (siehe Abschnitt 6.5.2.4) erzeugt.

6.6.3.5 Daten verifizieren

Wenn der Anwender die Datenverifikation ausgewählt hat, öffnet sich ein Dialog, in dem die zu prüfenden OpenPGP Nachrichten im Dateisystem selektiert werden können. Nach einem Klick auf den Schalter *OK*, wird die Verifikation über den `MessageParser` (siehe Abschnitt 6.5.2.3) durchgeführt. Ein Statusfenster informiert den Anwender über erfolgreiche und fehlgeschlagene Prüfungen.

6.6.3.6 Schlüssel importieren

Schlüssel können sowohl von einem Schlüsselserver, als auch aus einer Datei importiert werden. Über die Toolbar kann der Anwender zwischen beiden Optionen wählen. Für den Fall, dass der Dateiimport selektiert wurde, öffnet sich ein Dialog, der die Auswahl der zu importierenden Schlüssel erlaubt. Bestätigt der Anwender die Auswahl werden die selektierten Schlüssel über den `MessageParser` (siehe Abschnitt 6.5.2.3) eingelesen und in der Datenbank gespeichert.

Der Import über einen Schlüsselserver öffnet den `ImportPublicKeyWizard`, der nur die Seite `ImportPublicKeyPage1` enthält. Der Anwender muss zunächst einen der verfügbaren Schlüsselserver auswählen. Die Liste der Schlüsselserver wird über die Benutzerpräferenzen (siehe Abschnitt 6.6.4) verwaltet und ist momentan statisch. Danach kann über das Suchfeld nach einem Schlüssel über ID, Fingerabdruck oder einen Teil der Benutzerkennung gesucht werden. Die Trefferliste wird im unteren Bereich angezeigt. Bild 6.10 zeigt beispielsweise die Ergebnisliste für den Suchbegriff *TestMe*. Nachdem der Anwender einen oder mehrere Schlüssel selektiert und den Wizard abgeschlossen hat, werden die Schlüssel importiert und in der Datenbank gespeichert.

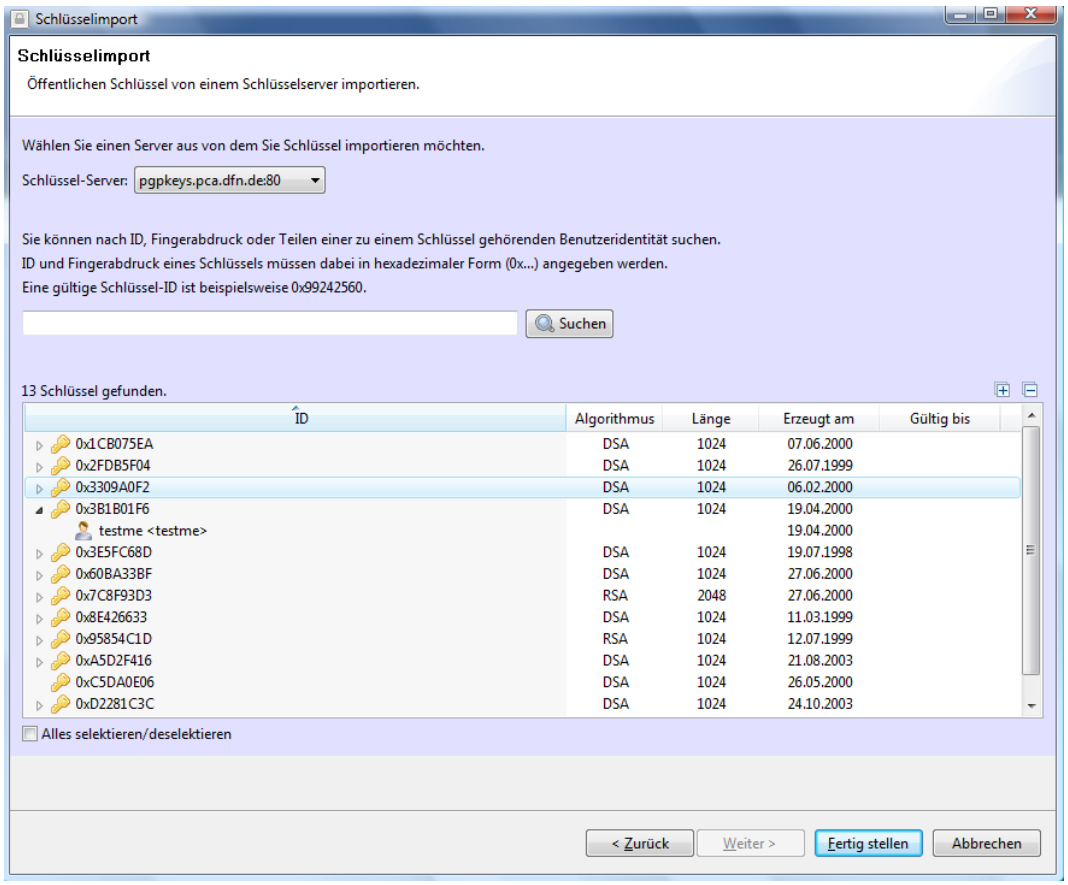
Schlüsselimport
Schlüsselimport
Öffentlichen Schlüssel von einem Schlüsselserver importieren.
Wählen Sie einen Server aus von dem Sie Schlüssel importieren möchten.
Schlüssel-Server: pgpkeys.pca.dfn.de:80
Sie können nach ID, Fingerabdruck oder Teilen einer zu einem Schlüssel gehörenden Benutzeridentität suchen.
ID und Fingerabdruck eines Schlüssels müssen dabei in hexadezimaler Form (0x...) angegeben werden.
Eine gültige Schlüssel-ID ist beispielsweise 0x99242560.
Suchen
13 Schlüssel gefunden.
ID Algorithmus Länge Erzeugt am Gültig bis
0x1CB075EA DSA 1024 07.06.2000
0x2FDB5F04 DSA 1024 26.07.1999
0x3309A0F2 DSA 1024 06.02.2000
0x3B1B01F6 DSA 1024 19.04.2000
testme <testme> 19.04.2000
0x3E5FC68D DSA 1024 19.07.1998
0x60BA33BF DSA 1024 27.06.2000
0x7C8F93D3 RSA 2048 27.06.2000
0x8E426633 DSA 1024 11.03.1999
0x95854C1D RSA 1024 12.07.1999
0xA5D2F416 DSA 1024 21.08.2003
0xC5DA0E06 DSA 1024 26.05.2000
0xD2281C3C DSA 1024 24.10.2003
Alles selektieren/deselektieren
< Zurück
Weiter >
Fertig stellen
Abbrechen

Abbildung 6.10: Schlüsselimport über einen Schlüsselserver

6.6.3.7 Schlüssel exportieren

Die Benutzeroberfläche unterstützt bis dato nur den Export von Schlüsseln in eine Datei. Der Anwender kann aus einer Liste die zu exportierenden Schlüssel, sowie das Dateiausgabeformat (binär oder ascii-armored) und den Speicherort wählen.

6.6.3.8 Sprachauswahl

Momentan stehen die in der Benutzeroberfläche angezeigten Text in deutscher und englischer Sprache zur Verfügung. Eine Auswahl überschreibt die aktuelle Sprach-Benutzerpräferenz (siehe nächster Abschnitt) und startet die Anwendung neu, damit die Änderungen wirksam werden.

6.6.4 Benutzereinstellungen

Eclipse erlaubt die Speicherung von Benutzereinstellungen über mehrere Sitzungen hinweg. Diese werden in einem sogenannten `PreferenceStore` in der Datei `.metadata/.plugins/org.eclipse.core.runtime/.settings/de.tud.cdc.openpgp.ui.prefs` gespei-

chert, die innerhalb des Workspace liegt. Es handelt sich dabei um eine Propertydatei, die die Benutzereinstellungen als Schlüssel/Wert-Paare speichert. Der Schlüssel identifiziert eindeutig eine bestimmte Benutzereinstellung. Die vom OpenPGP Client verwalteten Präferenzen liegen im Paket `de.tud.cdc.openpgp.ui.preferences`. Aktuell gibt es zwei Präferenzseiten, `LanguagePage` und `KeyServerPage`. Die Klasse `LanguagePage` erlaubt über eine aufklappbare Liste (DropDown-List) die Auswahl der, in der Oberfläche verwendeten Sprache. Nach einer Änderung der Sprache, wird die Anwendung automatisch neu gestartet, da die von Eclipse verwendete Locale zur Laufzeit nicht geändert werden kann. Diese Aufgabe wird von der Hilfsklasse `Languages` übernommen. `KeyServerPage` verwaltet eine Liste von Schlüsselservern, von denen öffentliche Schlüssel importiert bzw. exportiert werden können. Die Liste der Sprachen, sowie die Liste der Schlüsselserver kann allerdings im Moment noch nicht editiert werden.

Die Standardeinstellungen für beide Präferenzseiten werden innerhalb der Klasse `Preferences` gesetzt, die als Initialisierungsklasse (abstract initializer) innerhalb der `plugin.xml` registriert wird. Damit beide Klassen als Präferenzseiten identifiziert und von der Workbench automatisch den Benutzereinstellungen hinzugefügt werden, erweitern sie den Extension Point `org.eclipse.ui.preferencePages` und implementieren das von Eclipse vorgegebene Interface `IWorkbenchPreferencePage`. Beide Präferenzseiten sind in Abbildung 6.11 dargestellt.

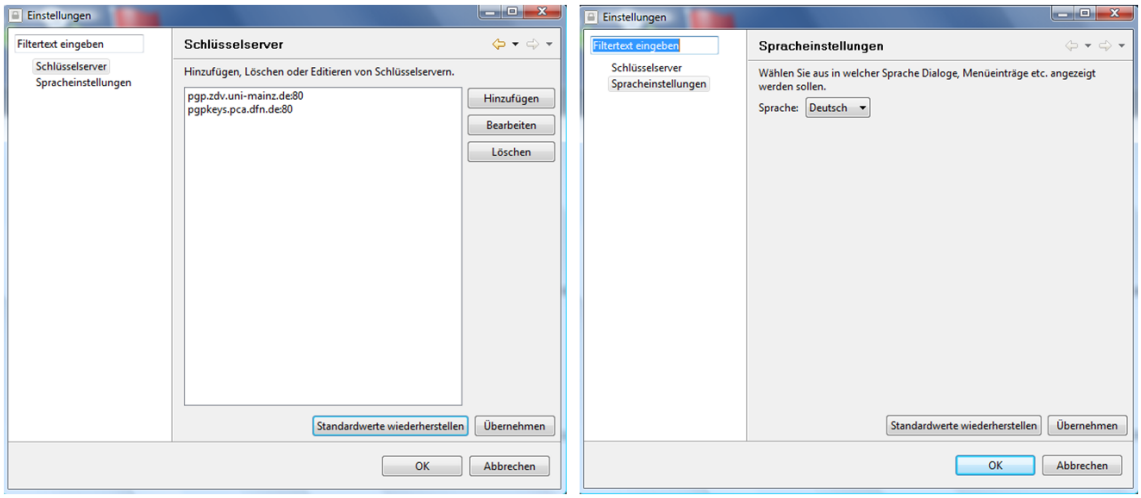

Abbildung 6.11: Präferenzseiten in den Benutzereinstellungen

6.7 Benutzerhandbuch

Eine sehr einfach gehaltene Vorlage für ein Benutzerhandbuch befindet sich im Plug-in `de.tud.cdc.openpgp.help`. Damit das Benutzerhandbuch automatisch dem Hilfesystem hinzugefügt wird, erweitert das Plug-in den Extension Point `org.eclipse.help.toc`. Die Vorlage enthält noch keinen Inhalt, sondern demonstriert stattdessen wie

Hilfeinhalte in die Anwendung integriert werden können. Als Vorlage wurde die Eclipse Standardhilfeseite verwendet. Die Vorlage sieht die Verwendung von mehrsprachigen Hilfetexten vor. Diese müssen unterhalb des *nl*-Ordners liegen, damit Eclipse zur Laufzeit, anhand der eingestellten Locale bestimmen kann, welches Benutzerhandbuch angezeigt werden soll[135]. Dementsprechend liegen im *de*-Ordner unterhalb vom *nl*-Verzeichnis die Vorlagen für das deutsche, im *en*-Ordner die Vorlagen für das englische Handbuch. Die Struktur des angezeigten Inhaltsverzeichnisses wird in den Dateien *testToc.xml* und *toc.xml* festgelegt, die eigentlichen Seiten des Benutzerhandbuchs liegen als HTML-Seiten im Unterordner *html*.

6.8 Internationalisierung

Eine Anforderung an den OpenPGP Client ist die Mehrsprachigkeit der Anwendung. Die in der Benutzeroberfläche dargestellten Texte sollen zunächst in deutscher und englischer, später gegebenenfalls auch in anderen Sprachen verfügbar sein. Eclipse verwendet für die Internationalisierung den Standardmechanismus von Java. Die zu lokalisierenden Texte werden dabei in *Property-Resource-Bundles*, im konkreten Fall in Property-Dateien, abgelegt. Dabei handelt es sich um Textdateien, die sogenannte *Schlüssel/Wert-Paare* (key/value pairs) enthalten. Der Schlüssel identifiziert dabei den zu lokalisierenden Text, der Text selbst wird im Wert gespeichert[136]. Eclipse stellt hierfür den sogenannten *Externalisierungsmechanismus* zur Verfügung, der Entwickler bei der Erstellung von Resource-Bundles unterstützt.

Ein Bundle wird über einen Namen und eine bestimmte Locale identifiziert, so enthält beispielsweise die Datei `message_de.properties` mit der Localeangabe *de* Texte in deutscher Sprache, während `message_en.properties` Texte in englischer Sprache enthält. Wenn Eclipse für die aktuelle Locale kein passendes Bundle findet, so wird standardmäßig nach einem Bundle ohne Locale gesucht, für das obige Beispiel wäre der Dateiname demnach `message.properties`.

Der OpenPGP Client verwendet pro Verzeichnis, in dem sich Klassen mit zu lokalisierenden Texten befinden, je ein Bundle für deutsche und ein Bundle für englische Texte. Die Übersetzungen für das Core-Plug-in sowie für das UI-Plug-in wurden in die Fragment-Plug-ins `de.tud.cdc.openpgp.core.nl` und `de.tud.cdc.openpgp.ui.nl` ausgelagert. So können später weitere Übersetzungen hinzugefügt werden, die dann automatisch mit dem Host-Plug-in zusammengeführt werden. Die Bundles ohne Localeangabe (siehe

[135] Die Dateien des Benutzerhandbuchs können zusätzlich auch auf oberster Ebene im Plug-in abgelegt werden. Findet Eclipse unterhalb des *nl*-Ordners kein zur Locale passendes Verzeichnis, werden diese Dateien für die Darstellung benutzt.

[136] Prinzipiell erlauben Resource-Bundles die Speicherung eines beliebigen Java-Objekts. Weiterhin müssen die Texte nicht unbedingt in einer Property-Datei gespeichert werden. Siehe hierzu auch http://java.sun.com/developer/technicalArticles/Intl/ResourceBundles/ oder http://java.sun.com/javase/6/docs/api/java/util/ResourceBundle.html

oben) sind weiterhin im Host-Plug-in enthalten, damit auch ohne Hinzufügen des jeweiligen Fragments Texte vorhanden sind. Standardmäßig enthalten diese Bundles die Texte in englischer Sprache.

Neben dem Core- und dem UI-Plug-in, welche die eigentliche OpenPGP Client Funktionalität enthalten, müssen auch Übersetzungen für die Texte der verwendeten Eclipse-Komponenten (Workbench, Help-Dialog etc.) bereitgestellt werden. Eclipse wird in englischer Sprache ausgeliefert. Bis zur Eclipse Version 2.1 wurden daher von IBM sogenannte *Language-Packs* zur Verfügung gestellt, die Übersetzungstexte in diversen Sprachen enthielten. Für die im OpenPGP Client verwendete Eclipse Version 3.3 gibt es keine offiziellen Language Packs mehr[137] und die Language Packs der früheren Version sind aufgrund von Änderungen und Erweiterungen in der neuen Version nur bedingt einsetzbar. Um dennoch eine Mehrsprachigkeit der Eclipse Komponenten zu ermöglichen, wurden daher die von einigen freiwilligen Eclipse-Entwicklern, sowie vom Autor selbst erstellte Übersetzungs-Texte verwendet[138].

Diese befinden sich in den folgenden Plug-ins:

- org.eclipse.help.webapp.nl1

  Enthält die Übersetzungen für die Darstellung der OpenPGP Client Hilfe.

- org.eclipse.jface.nl1

  Enthält die Übersetzungen für die JFace-Komponenten.

- org.eclipse.workbench.nl1

  Enthält die Übersetzungen für die Workbench-Komponenten.

- org.eclipse.ui.views.log

  Enthält die Übersetzungen für den Error Log View (Siehe Abschnitt 6.9)

Hierbei ist zu beachten, dass entsprechend den Anforderungen an den OpenPGP Client, nur die englischen und deutschen Übersetzungen vollständig sind.

6.9 Fehlerprotokollierung

Um während der Verwendung des OpenPGP Clients auftretende Fehler, Warnungen sowie allgemeine Statusinformationen im nachhinein analysieren zu können, werden diese in einer Datei protokolliert. Dieser Vorgang wird auch als *Logging* bezeichnet.

[137] Zur Zeit ist das Projekt Babel in Vorbereitung, dass sich dediziert mit der Problematik der Internationalisierung von Eclipse-Anwendungen beschäftigen soll. Siehe hierzu http://www.eclipse.org/babel/

[138] Siehe hierzu auch die Copyright-Informationen in den jeweiligen Property-Dateien.

Der OpenPGP Client verwendet den in Eclipse integrierten Loggingmechanismus, der über die Schnittstelle `org.eclipse.core.runtime.ILog` angesprochen werden kann. Die Verwendung von ILog bietet, neben der Standard-Logging-Funktionalität, den Vorteil, dass die Protokolleinträge in einem speziellen, von Eclipse zur Verfügung gestellten View angezeigt werden können (siehe unten). Einen Ansatz für die Einbindung von alternativen Logging-Frameworks wie *log4j*[139] oder beispielsweise das im Java Development Kit (JDK) integrierte Logging[140] beschreibt der Artikel von John Franey[141].

ILog definiert die Methode `log(org.eclipse.core.runtime.IStatus status)` die ein Statusobjekt protokolliert. Das Statusobjekt enthält nähere Informationen zum aufgetretenen Ereignis.

Dazu zählen:

- Statustyp
  Der Statustyp gibt an, ob es sich um eine Warnung, Information, Fehler etc. handelt.
- Plug-in ID
  Die Plug-in ID ist der eindeutige Bezeichner des Plug-ins, welches das Statusobjekt erzeugt.
- Nachricht
  Eine Nachricht, die das aufgetretene Ereignis beschreibt.
- Exception
  Wenn ein Fehler aufgetreten ist, kann dem Status-Objekt das zugrundeliegende Exception-Objekt übergeben werden.

Um den Zugriff auf die ILog-Schnittstelle zentral zu halten, wurden die Plug-in Activator-Klassen `de.tud.cdc.openpgp.core.CorePlugin` und `de.tud.cdc.openpgp.UIActivator` um die folgenden Methoden ergänzt:

- `logError(String msg, Exception e)`
- `logInfo(String msg)`
- `logWarning(String msg)`

Diese erzeugen die Status-Objekte, die für die Protokollierung notwendig sind und ermöglichen somit anderen Klassen innerhalb der Plug-ins einen vereinfachten Zugriff auf die Logging-Funktionalität. Die generierte Protokolldatei *.log* liegt unterhalb des Installationsverzeichnis im Ordner *.metadata*.

[139] Siehe http://logging.apache.org/log4j/
[140] Siehe http://java.sun.com/javase/6/docs/api/java/util/logging/package-summary.html
[141] Siehe [Fra05]

Für die Darstellung der Protokolleinträge wird der Eclipse Standard Error Log-View `org.eclipse.ui.internal.views.log.LogView` verwendet, der automatisch die Liste aller Einträge aus der Log-Datei ausliest. Dies ist beispielhaft in Abbildung 6.12 zu sehen. Per Doppelklick lassen sich die Details eines einzelnen Eintrags in einem separaten Fenster öffnen (siehe Abbildung 6.13). Der View wird, bis auf das Hinzufügen der deutschen Übersetzungen für die UI-Elemente (siehe Abschnitt 6.8), in unveränderter Form genutzt.

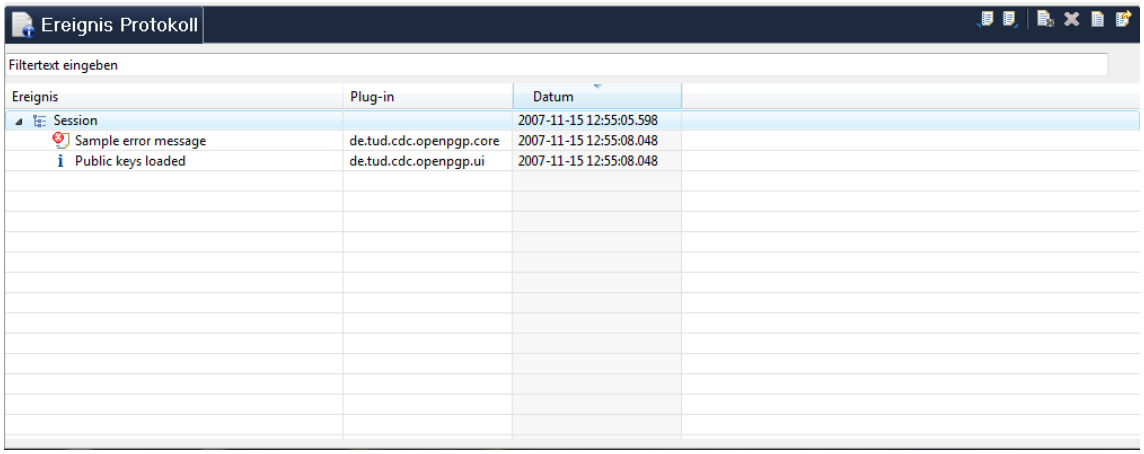

Abbildung 6.12: Darstellung von Einträgen aus dem Fehlerprotokoll im Error Log View

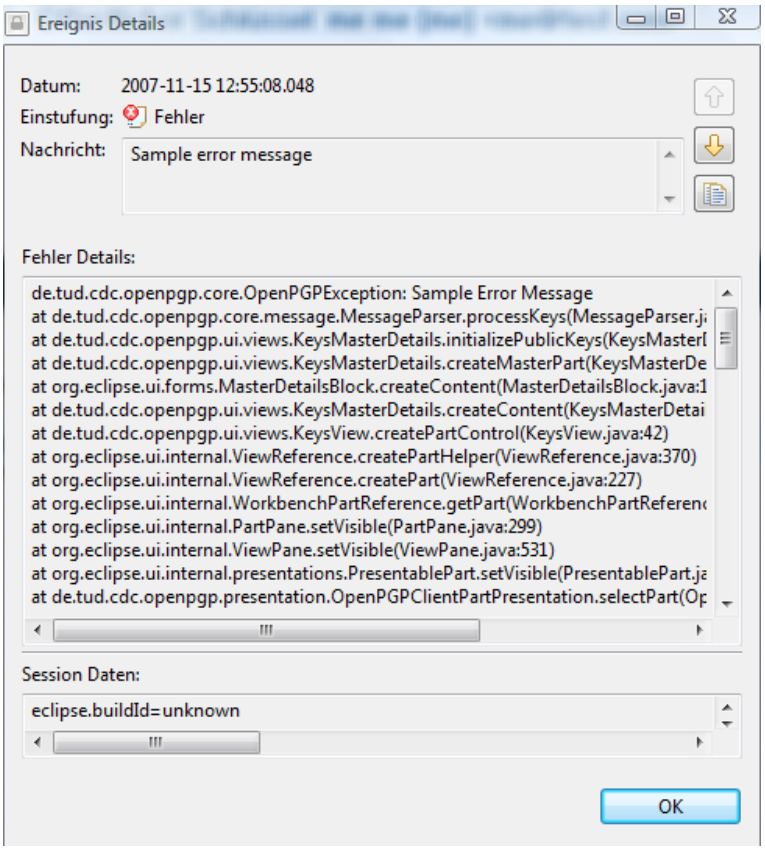

Abbildung 6.13: Details eines Fehlerprotokoll-Eintrags

Kapitel 7

Zusammenfassung

Im Rahmen dieser Arbeit wurde das Design und die prototypische Implementierung eines OpenPGP-Clients unter Verwendung von Java und dem Eclipse RCP Framework dargestellt. Die entstandene Software basiert auf einer unvollständigen, am Fachgebiet erstellten OpenPGP Bibliothek, die im Zuge der Entwicklung um zahlreiche Funktionen (siehe Kapitel 6) erweitert wurde.

Die erstellte Lösung ist aufgrund der hochgradig flexiblen Eclipse-Architektur leicht erweiterbar. Das bestehende `UI`-Plug-in lässt sich um neue Aktionen, Sichten oder Editoren etc. ergänzen, die Modifikationen und Erweiterungen der OpenPGP-Implementierung reflektieren und automatisch per Extension-Mechanismus in die RCP-Anwendung integriert werden. Gegebenfalls können auch neue Plugins erstellt werden, die die Anwendung um zusätzliche Funktionalität bereichern und nach Bedarf im laufenden Betrieb der Anwendung über den Update-Mechanismus nachinstalliert werden können[142].

Die OpenPGP-Bibliothek selbst ist durch die Verwendung einer abstrakten Klassenhierarchie so konzipiert worden, dass sie leicht an Änderungen des Nachrichten-Protokolls angepasst werden kann. So ist es z.B. problemlos möglich eine neue Paketversion in die bestehende Hierarchie zu integrieren. Zudem verfügt der OpenPGP-Client über eine moderne Benutzeroberfläche, die dem Anwender die grundlegenden kryptographischen Funktionen in übersichtlicher Form präsentiert, um auch Anwendern ohne fachlichen Hintergrund die Arbeit mit der Software zu erleichtern.

Der OpenPGP Client stellt daher meiner Meinung nach, eine gute Basis für weiterführende Projekte am Fachgebiet dar.

[142]Der Updatemechanismus wird in der aktuellen Version des OpenPGP Clients nicht unterstützt.

Literaturverzeichnis

[Ani07] Aniszczyk, Chris: *Revitalize your applications with Eclipse Forms*. http://www.ibm.com/developerworks/opensource/library/os-eclipse-forms/, 2007.

[AR96] Abdul-Rahman, Alfarez: *The PGP Trust Model*, 1996.

[Ash99] Ashley, John Michael: *The GNU Privacy Handbook*. http://www.gnupg.org/gph/en/manual.pdf, 1999.

[BNS05] Beutelspacher, Albrecht, Heike B. Neumann und Thomas Schwarzpaul: *Kryptografie in Theorie und Praxis*. Vieweg, 1. Auflage, Januar 2005.

[Buc03] Buchmann, Johannes: *Einführung in die Kryptographie*. Springer, 3. Auflage, September 2003.

[Buc05] Buchmann, Johannes: *Skript zur Vorlesung Public-Key-Infrastrukturen*. http://www.cdc.informatik.tu-darmstadt.de/lehre/SS05/vorlesung/PKI/resources.html, 2005.

[CDF+07] Callas, J., L. Donnerhacke, H. Finney, D. Shaw und R. Thayer: *RFC4880 - OpenPGP Message Format*. www.ietf.org/rfc/rfc4880.txt, November 2007.

[CLRS04] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest und Clifford Stein: *Introductions to Algorithms*. MIT Press, 2. Auflage, Juni 2004.

[Col87] Colbourn, Charles J.: *The combinatorics of network reliability*. Oxford Univ. Press, 1987.

[CR04] Clayberg, Eric und Dan Rubel: *Eclipse. Building Commercial-Quality Plug-Ins*. Addison-Wesley, Juni 2004.

[Del06] Delap, Scott: *Understanding how Eclipse plug-ins work with OSGi*. http://www.ibm.com/developerworks/library/os-ecl-osgi/index.html, 2006.

[Die00] Diestel, Reinhard: *Graphentheorie*. Springer, 2. Auflage, 2000.

[Eck04] Eckert, Claudia: *IT-Sicherheit*. Oldenburg, 3. Auflage, 2004.

[ecl] ECLIPSE.ORG: *Eclipse Dokumentation*. `http://http://www.eclipse.org/documentation/`.

[FFS04] FREEMAN, ERIC, ELISABETH FREEMAN und KATHY SIERRA: *Head First Design Patterns*. O'Reilly, November 2004.

[Fle06] FLECK, MATTHIAS: *Zur Benutzbarkeit von PKI-Anwendungen im praktischen Einsatz*. Diplomarbeit, Technische Universität Darmstadt, Juni 2006.

[Fra05] FRANEY, JOHN: *Universal Logger Plug-ins for RCP Applications*. `http://www.eclipsezone.com/articles/franey-logging/?source=archives`, Dezember 2005.

[GA07] GALLARDO, DAVID und CHRIS ANISZCZYK: *Get started with the Eclipse Platform*. `http://www.ibm.com/developerworks/opensource/library/os-eclipse-platform/?S\_TACT=105AGX59\&S\_CMP=GR\&ca=dgr-eclipse-1`, Juli 2007.

[Gar96] GARFINKEL, SIMON: *PGP: Pretty Good Privacy: Verschlüsselung von E-Mail*. O'Reilly/International Thompson Verlag, 1996.

[GB03] GAMMA, ERICH und KENT BECK: *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison Wesley, 2003.

[GHJ04] GAMMA, ERICH, RICHARD HELM und RALPH E. JOHNSON: *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, Juli 2004.

[Glo05] GLOZIC, DEJAN: *Eclipse Forms: Rich UI for the Rich Client*. `http://www.eclipse.org/articles/Article-Forms/article.html`, 2005.

[Gri02] GRINDSTAFF, CHRIS: *How to use the JFace Tree Viewer*. `http://www.eclipse.org/articles/Article-TreeViewer/TreeViewerArticle.htm`, 2002.

[IBM06] IBM: *Eclipse Platform Technical Overview*. `http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.pdf`, 2006.

[JWH06] JONCZY, JACEK, MARKUS WÜTHRICH und ROLF HAENNI: *A Probabilistic Trust Model for GnuPG*. `www.iam.unibe.ch/~jonczy/papers/JWH06.pdf`, 2006.

[Keh02] KEHN, DAN: *How to Test Your Internationalized Eclipse Plug-In*. `http://www.eclipse.org/articles/Article-TVT/how2TestI18n.html`, August 2002.

[KFL02] Kehn, Dan, Scott Fairbrother und Cam-Thu Le: *How to Internationalize your Eclipse Plug-In*. `http://www.eclipse.org/articles/Article-Internationalization/how2I18n.html`, August 2002.

[Knu98] Knudsen, Jonathan: *Java Cryptography*. O'Reilly, Mai 1998.

[MF06] Moreira, Jorge Castiñeira und Patrick Guy Farrell: *Essentials of Error-Control Coding*. Wiley, 2006.

[ML05] McAffer, Jeff und Jean-Michel Lemieux: *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications*. Addison Wesley, Oktober 2005.

[NW04] Northover, Steve und Mike Wilson: *SWT: The Standard Widget Toolkit, Volume 1*. Addison-Wesley, Juni 2004.

[Sch06] Schneier, Bruce: *Angewandte Kryptographie*. Pearson Studium, 2006.

[Sha03] Shaw, David: *The OpenPGP HTTP Keyserver Protocol (HKP)*. `http://ietfreport.isoc.org/all-ids/draft-shaw-openpgp-hkp-00.txt`, 2003.

[SM02] Sun Microsystems, Inc.: *Core J2EE Patterns - Data Access Object*. `http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html`, 2002.

[WT98] Whitten, Alma und J. D. Tygar: *Usability of Security: A Case Study*. Technischer Bericht, Carnegie Mellon University School of Computer Science, Dezember 1998.